HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
René Böttcher

Tablet-Umsetzung der App HSMWmobil

Mittweida, 2013

Fakultät Elektro- und Informationstechnik

BACHELORARBEIT

Tablet-Umsetzung der App HSMWmobil

Autor:
Herr René Böttcher

Studiengang:
Multimediatechnik

Seminargruppe:
MK10w1-B

Erstprüfer:
Prof. Dr.-Ing. Frank Zimmer

Zweitprüfer:
M.Sc. Rico Thomanek

Einreichung:
Mittweida, 26.07.2013

Verteidigung/Bewertung:
Mittweida, 2013

Faculty Elektro- und Informationstechnik

BACHELOR THESIS

Tablet-Umsetzung der App HSMWmobil

author:
Mr. René Böttcher

course of studies:
Multimediatechnik

seminar group:
MK10w1-B

first examiner:
Prof. Dr.-Ing. Frank Zimmer

second examiner:
M.Sc. Rico Thomanek

submission:
Mittweida, 26.07.2013

defence/ evaluation:
Mittweida, 2013

Bibliografische Beschreibung:

Böttcher, René:
Tablet-Umsetzung der App HSMWmobil. – 2013. – 6, 63, 1 S.
Mittweida, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik, Bachelorarbeit, 2013

Referat:

Die vorliegende Arbeit befasst sich mit der Umsetzung von Anpassungen und Optimierungen der bestehenden App der Hochschule Mittweida für die Gerätegruppe der Tablets. Das Hauptziel ist die Funktionalitäten der bestehenden App zu erhalten und das zusätzliche Platzangebot auf der genannten Gerätegruppe effizient zu nutzen. Im Blickpunkt der Arbeit steht außerdem die Gestaltung und Optimierung der bestehenden Benutzeroberfläche. Die App soll sowohl für die Gerätegruppe der Smartphones, als auch für Tablets zur Verfügung stehen.

Inhalt

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADT	Android Development Toolkit
API	Application Programming Interface
APK	Application Package File
AVD	Android Virtual Device
dp	Density-independent Pixels
dpi	Dots per Inch
FIT	Frequent Important Typical
IDC	International Data Corporation
MTP	Media Transfer Protocol
OS	Operating System
PTP	Picture Transfer Protocol
SDK	Software Development Kit
SVN	Subversion
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1 Einleitung

Das Betriebssystem und dessen beinhaltende Plattform Android, welche von der Open Handset Alliance, mit deren Hauptentwickler Google Inc., entwickelt wird, wurde am 21. Oktober 2008 erstmals vorgestellt. Der Erfolg von Android war zu diesem Zeitpunkt nicht absehbar, da Apple bereits mit dem iPhone und dessen Betriebssystem iOS den Markt dominierte. Des Weiteren war der Markt für mobile Betriebssysteme bereits stark entwickelt, so dass neben Apples iOS bekannte Betriebssystem wie Symbian OS von Nokia, Windows Mobile von Microsoft oder Blackberry OS von RIM den Markt bestimmten.

Angetrieben durch das Unternehmen Google Inc., welches das Betriebssystem hauptsächlich entwickelt und vertreibt, stieg der Marktanteil von Android seit dem Jahr 2008 rapide und beansprucht seit dem vierten Quartal 2010 die Spitzenposition für sich. Seitdem dominiert Google mit Android den Markt nach Belieben, sodass Android nach aktuellen Statistiken von dem internationalen Marktforschungsunternehmen Gartner etwa 75% des Marktes für mobile Betriebssysteme weltweit[1] beherrscht. Zusammen mit iOS beansprucht Android laut IDC[2] einen Gesamtmarktanteil von 92,3% im Mai 2013.

Worldwide smart phone market
Market shares Q4 2010, Q4 2009

OS vendor	Q4 2010 shipments (millions)	% share	Q4 2009 shipments (millions)	% share	Growth Q4'10/Q4'09
Total	101.2	100.0%	53.7	100.0%	88.6%
Google*	33.3	32.9%	4.7	8.7%	615.1%
Nokia	31.0	30.6%	23.9	44.4%	30.0%
Apple	16.2	16.0%	8.7	16.3%	85.9%
RIM	14.6	14.4%	10.7	20.0%	36.0%
Microsoft	3.1	3.1%	3.9	7.2%	-20.3%
Others	3.0	2.9%	1.8	3.4%	64.8%

*Note: The Google numbers in this table relate to Android, as well as the OMS and Tapas platform variants
Source: Canalys estimates, © Canalys 2011

Abbildung 1 Statistik durch Canalys[3] für die Verteilung mobiler Betriebssysteme 2011

[1] Vgl.: http://www.zdnet.de/88154889/gartner-android-erreicht-fast-75-prozent-marktanteil/, 2013.

[2] Vgl.: http://www.idc.com/getdoc.jsp?containerId=prUS24108913, 2013.

[3] Vgl.: http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform, 2013.

Diese Rechnungen werden durch den Smartphone-Markt geschönt und Android versucht seit Version 3.0 auch im Tablet-Markt Fuß zu fassen. Dieser Markt, welcher durch das iPad von Apple geschaffen wurde, wurde lange Zeit durch Apple selbst beherrscht. Im ersten Quartal dieses Jahres konnte Android erstmals die Dominanz brechen und die Spitzenposition für sich behaupten.

Top Tablet Operating Systems, Shipments, and Market Share, 2013 Q1 (Shipments in Millions)

Vendor	1Q13 Unit Shipments	1Q13 Market Share	1Q12 Unit Shipments	1Q12 Market Share	Year-over-Year Growth
Android	27.8	56.5%	8.0	39.4%	247.5%
iOS	19.5	39.6%	11.8	58.1%	65.3%
Windows	1.6	3.3%	0.2	1.0%	700.0%
Windows RT	0.2	0.4%	0.0	N/A	N/A
Others	0.1	0.2%	0.2	1.0%	-50.0%
Total	**49.2**	**100.0%**	**20.3**	**100.0%**	**142.4%**

Abbildung 2 Statistik des IDC[4] für das erste Quartal 2013 des Marktes für Tablets

Für den Bereich der Smartphones existiert von der Hochschule Mittweida seit dem 17. Dezember 2012 die App HSMWmobil für Android und iOS in Version 1.0. Diese Version ist speziell auf die Größe der Smartphones optimiert und bietet Studenten, Mitarbeitern, sowie allen Angehörigen der Hochschule nützliche Informationen über das Hochschulleben. Dabei steht der Campus im Mittelpunkt, so dass es möglich ist, sich zu Gebäuden der Hochschule mittels des Campusplans navigieren zu lassen, nützliche Informationen zu Sprechzeiten, Telefonnummern und E-Mail-Adresse der Angehörigen der Hochschule über ein Kontaktbuch zu erhalten, interessante Nachrichten über die Hochschule aus dem News- oder Blogbereich zu lesen, mit Hilfe des Mensaplans den Speiseplan der Mensa für die aktuelle und folgende Woche einzusehen, den eigenen Stundenplan einzublenden und entsprechende Informationen zu Dozenten und Räumen zu erhalten, einen Radio-Stream des hochschuleigenen Radios Radio-Mittweida zu lauschen oder in der neusten Ausgabe der Novum auf dem Smartphone zu stöbern.

[4] Vgl.: http://m.androidcentral.com/idc-android-now-leads-tablet-market-565-share, 2013.

1.1 Problemstellung

Seit der Veröffentlichung der Version 1.0 von HSMWmobil hat sich bei der App für Android und iOS nur wenig getan. Es wurden kleinere Fehler behoben und entsprechende Updates veröffentlicht, die diese Version optimiert und fehlerbereinigt haben.

Hinter den Kulissen wurde jedoch eifrig weiter entwickelt und zum Beispiel das Modul der Notenanzeige integriert. Da die Version 1.0 sich zwar an die Richtlinien von Google für Android hält, jedoch seit Android 4.0 – Ice Cream Sandwich – von Google überarbeitete Richtlinien und Konzepte für das Benutzerempfinden und das Design existieren, wurde der Beschluss gefasst diese Richtlinien und Konzepte von Android 4.0 umzusetzen.

Des Weiteren wurde mit Beginn der Entwicklung beschlossen, die Apps für Android und iOS ebenfalls für Tablet-Geräte anzubieten und zu veröffentlichen. Dies wurde mit der Veröffentlichung von Version 1.0 auch umgesetzt, jedoch wurde die App sowohl auf Android, als auch auf iOS-Seite nicht explizit angepasst oder optimiert, so dass die Bedienung zweckmäßig vonstattenging. So kann die App nur im Portrait-Modus – der vertikalen Neigung des Endgerätes – und nicht im Landscape-Modus – der horizontalen Neigung des Endgerätes – genutzt werden. Des Weiteren wirkt die App kahl und trist, da der zusätzliche Platz nicht genutzt wird und die Inhalte lediglich auf Smartphone-Größe dargestellt werden.

Mit der gestiegenen Verbreitung von Tablets in den vergangenen Jahren und den damit verbundenen neuen Anforderungen an bestehende Apps, ist ein Umdenken im Bereich der App-Entwicklung gefordert. Die Möglichkeit die App für verschiedene Plattformen – die Smartphones und Tablets – anzubieten, erfordern die Ausarbeitung neuer Konzepte, Anpassungen und Optimierungen für bestehende Apps in vielerlei Hinsicht. So ist es notwendig die bestehenden Konzepte zu Überdenken und im Bereich der Tablet-Entwicklung tätig zu werden, um den Anforderungen gerecht werden zu können und das zusätzliche Platzangebot sinnvoll und strukturiert zu nutzen.

Diese Anforderungen sollen im Laufe des Bachelor-Projektes bedacht und umgesetzt werden. Im Zuge des Bearbeitungszeitraumes müssen verschiedene Phasen durchlaufen werden, damit die neuen Richtlinien und Konzepte umgesetzt werden können. So müssen zum einen die Voraussetzungen für die Umsetzungen entsprechend geschaffen, und zum anderen die Konzepte der Tablet-Optimierung umgesetzt werden.

1.2 Motivation

Die Wahl des Themas für das Bachelor-Projekt fiel auf die Umsetzung einer Tablet-Version der hochschuleigenen App HSMWmobil und befasst sich mit einem aktuellen medialen Thema. Die Verbreitung von mobilen Geräten wie Smartphones und Tablets, sowie Apps für diese Geräte hat in den letzten Jahren stark zugenommen. Des Weiteren ist ein Trend erkennbar, der diese Entwicklung fortsetzt, wodurch es als Entwickler immer wichtiger wird schnell und gezielt Apps weiterzuentwickeln und diese auf so vielen Plattformen wie möglich zugänglich zu machen und zu halten.

Als einer von drei Entwicklern der App HSMWmobil für Android und als interessierter Beobachter des Marktes für mobile Geräte, war es mein persönlicher Wunsch an der Umsetzung für Tablets arbeiten zu können. Der Wunsch ist bedingt durch das Arbeiten im Bereich der Android-Entwicklung, des großen Interesses an und um Android, sowie der Weiterführung der Arbeit an der App der Hochschule Mittweida. Daher war es naheliegend auch für das Bachelor-Projekt ein Thema zu definieren, welches sich mit Android, der Entwicklung von HSMWmobil und einem neuen Themenbereich der Entwicklung befasst.

Durch die Erarbeitung und Umsetzung der Konzepte zur Optimierung ist es möglich die Kenntnisse im Bereich der Software- und Android-Entwicklung zu festigen und zu vertiefen, was ebenfalls für die Wahl des Themas entscheidend war. Des Weiteren werden diese Kenntnisse um ein sehr interessantes und umfangreiches Gebiet der Android-Entwicklung erweitert. Die App der Hochschule Mittweida reift durch die Umsetzung ebenfalls in ihrer eigenen Entwicklung. Viele Bestandteile des Quelltextes werden dabei überarbeitet und somit können in diesem Zuge Verbesserungen in die Entwicklung einfließen. Des Weiteren wird HSMWmobil für eine neue Gerätegruppe umgesetzt und ist nach Abschluss des Projektes dort präsent.

1.3 Kapitelübersicht

Die Bachelorarbeit besteht aus fünf ausführlichen Kapiteln.

Nach der allgemeinen Einleitung des ersten Kapitels werden im **Kapitel 2** die Rahmenbedingungen und Voraussetzungen erläutert. Neben der Ausgangssituation, Zielsetzung und Anforderungen, stehen das Projektmanagement, sowie die Entwicklungs- und Testumgebung im Vordergrund.

Im Anschluss daran werden im **Kapitel 3** die grundlegenden Konzepte für die spätere Umsetzung beschrieben und erläutert.

Hinterher wird im **Kapitel 4** die Implementierung ausführlich dargestellt. Die Implementierung selbst befasst sich mit der Umsetzung der Anforderungen, den Anpassungen und Optimierungen, sowie den abschließenden Tests.

Schlussendlich wird im **Kapitel 5** ein Ausblick auf mögliche Weiterentwicklungen gegeben und Konzepte vorgestellt. Des Weiteren wird die Bachelorarbeit reflektiert und ein Fazit über die Arbeit gezogen.

2 Rahmenbedingungen

2.1 Ausgangssituation

Zu Beginn des Bearbeitungszeitraumes der Bachelorarbeit befindet sich die Version 1.0 in Google Play von Google, sowie im App Store von Apple.

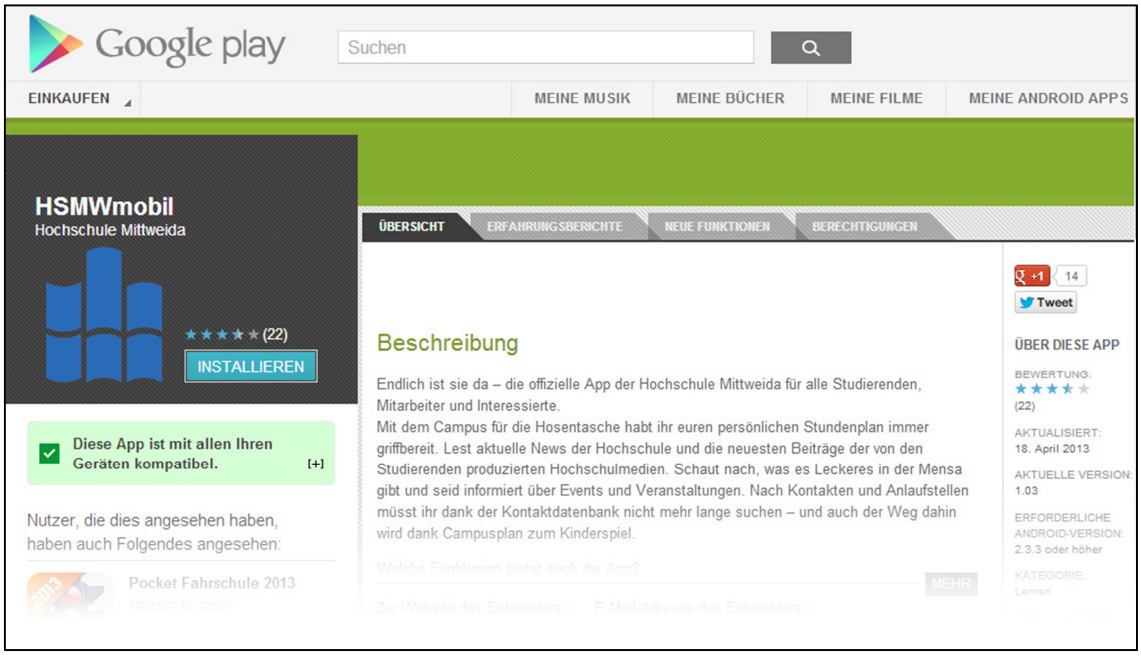

Abbildung 3 HSMWmobil in Google Play[5]

Der Funktionsumfang der jeweiligen Apps ist sowohl bei der App in Google Play, als auch im App Store von Apple identisch. Dabei greifen beide Apps auf identische Webservices und Daten zurück, um die Inhalte auf den Bildschirmen der mobilen Begleiter darzustellen.

Die Beschreibung und Funktionsliste in den jeweiligen Repositorien wurde von der Marketing-Abteilung der Hochschule Mittweida verfasst und veröffentlicht. Die Bildschirmfotos und Videos in Google Play und dem App Store stammen von den Entwicklungsgeräten der jeweiligen Plattform und wurden von den Entwicklern selbst hochgeladen. Das App-Icon der Android-Version wurde ebenfalls durch die Entwickler an die Plattform-Vorgaben von Google angepasst. Innerhalb des App Stores wird das App-Icon, wie auch in der App selbst, mit einem Leuchtschein versehen.

[5] Vgl.: https://play.google.com/store/apps/details?id=de.hsmw.app, 2013

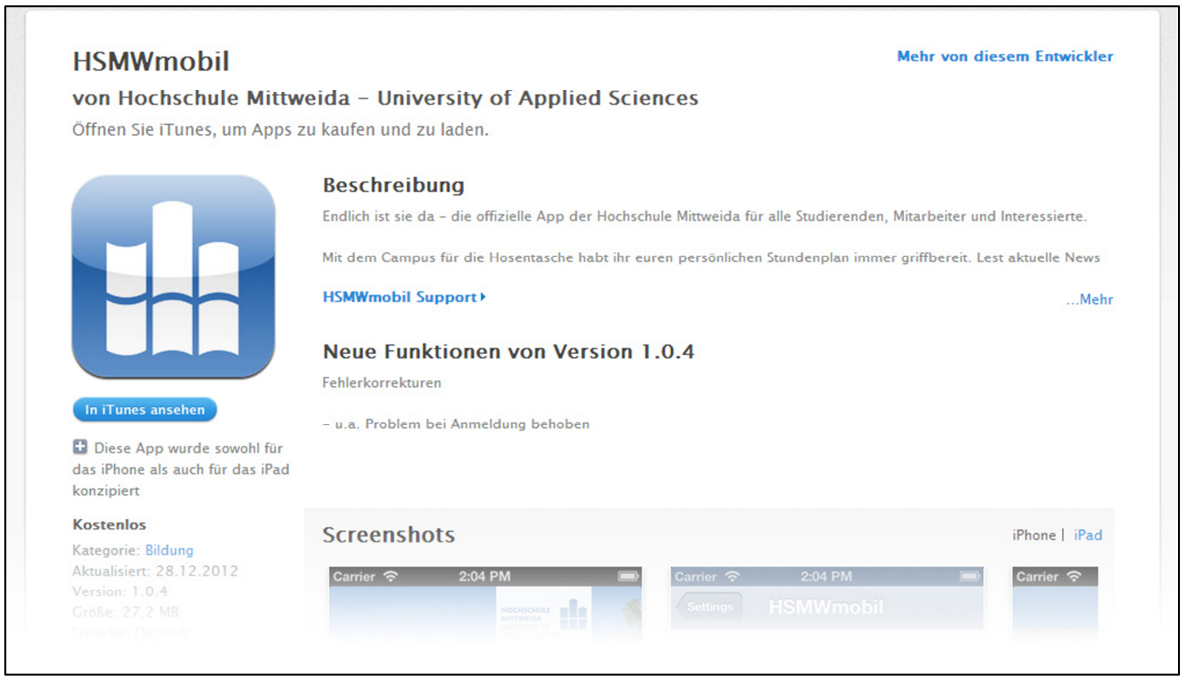

Abbildung 4 HSMWmobil im App Store[6]

Die Android-App wurde etwa 1100-aml aus Google Play heraus installiert und wird derzeit von etwa 900 Benutzern verwendet. Auf Grund anhaltender Probleme mit dem zugrundeliegenden Webservice hat die App in Google Play eine durchschnittliche Bewertung von 3,5 von 5 möglichen Sternen.

Abbildung 5 Developer Console[7] von Android für Entwickler

[6] Vgl.: https://itunes.apple.com/de/app/hsmwmobil/id581491936?mt=8, 2013

[7] Vgl.: https://play.google.com/apps/publish/?dev_acc=12464870337877015015&pli=1#AppListPlace, 2013.

Um diesem Problem gegenzusteuern und die Benutzer von der App weiter zu überzeugen, konnten diese Probleme durch interne Umstrukturierungen am Webservice vorläufig behoben werden. Eine weitere intensive Arbeit an den Webservices soll eine nachhaltige Grundlage für die Entwicklung der einzelnen Apps für Android, iOS und auch Windows 8 schaffen.

Als Basis für das Bachelor-Projekt dient die Version 1.03 von HSMWmobil, die am 18. April 2013 den Weg in Google Play gefunden hat. Teile des Quelltextes des Bachelor-Projektes flossen dabei bereits früh in die Entwicklung der Version 1.1 von HSMWmobil mit ein. Das Update für Version 1.1, welches viele Neuerungen, wie die für die Umsetzung des Bachelor-Projektes notwendige Action Bar oder das Modul Notenanzeige, beinhaltet, steht bereits in den Startlöchern und soll am 30. Juni 2013 veröffentlicht werden. Weitere Verbesserungen betreffen dabei das Modul Radio-Mittweida, welche den Stream in einen eigenen Service auslagert und dem Benutzer somit die Möglichkeit gibt, die App oder auch andere Apps zu nutzen, während der Stream im Hintergrund weiter abgespielt wird. Auf Geräten mit aktuelleren Android-Versionen ist es zudem möglich über das Notification Center[8] den Stream anzuhalten oder zu beenden. Die Detailansicht von News und Blog haben ebenso einen neuen Anstrich erhalten, wie die App selbst. Eine der großen Neuerung betrifft die Umsetzung der Google Maps-API v2[9] im Campusplan. Dieser wurde dabei von Grund auf neu entwickelt und bietet dem Benutzer viele hilfreiche neue Funktionen, sowie ein natives Google Maps-Erlebnis.

Alle Module, als auch einige Teilmodule bestehen aus eigenen Activities, welche im Zuge des Bachelor-Projektes überarbeitet werden müssen und durch eine Activity je Modul, sowie die dazugehörigen Fragments, ersetzt werden sollen.

[8] Benachrichtigungs-Zentrale von Android.

[9] Version 2 der Google Maps-API.

2.2 Zielsetzung

Im Rahmen der Umsetzung des Bachelor-Projektes werden verschiedene Ziele bereits im Vorfeld definiert, welche sich in Primär- und Sekundärziele einteilen lassen. Zu den Primärzielen gehört, neben der Schaffung einer gemeinsamen Software-Grundlage für Smartphone und Tablet, die sinnvolle Integration der Tablet-Optimierungen in den Quelltext und die Steigerung des positiven Benutzerempfindens.

Die Sekundärziele lassen sich in folgende Kategorien gliedern:

- Softwaretechnische Ziele
- Benutzerbezogene Ziele
- Zukunftsorientierte Ziele

Da für das Bachelor-Projekt die Software beziehungsweise die App im Vordergrund steht, werden die softwaretechnischen Ziele zuerst aufgeführt. Zu diesen Zielen gehören unter anderem die reibungslose Integration von Updates der weiteren Entwickler, die Optimierung des Quelltextes, sowie die Behebung von Fehlern im Falle des Auftretens. Des Weiteren sollen die bestehenden Module bei Bedarf überarbeitet und für Tablets optimiert werden.

Die, für App-Entwickler im Vordergrund stehenden, benutzerbezogenen Ziele sollen dabei nicht vernachlässigt und deshalb stets beachtet werden. Zu diesen Zielen gehören die Ausschöpfung des zusätzlichen Platzangebotes auf Tablets, die Integration der Action Bar für eine bessere In-App-Navigation, sowie die Schaffung einer intuitiven Bedienung auf Tablets.

Um eine App auch zukünftig sinnvoll und effizient weiterentwickeln zu können, müssen innerhalb jedes Projektes verschiedene Anforderungen erfüllt werden. Diese Anforderungen müssen in einem Projekt somit auch als Ziel aufgeführt werden. Zu diesen Zielen gehören ein strukturierter, aufgeräumter und gut dokumentierter Quelltext, sowie die Umsetzung der Objektorientierten Programmierung und die ständige Optimierung des integrierten Quelltextes.

Am Ende der Bachelorarbeit sollen die Primärziele vollständig umgesetzt werden, sowie die Sekundärziele weitestgehend beachtet und umgesetzt worden sein. Für das Bachelor-Projekt ist es ebenfalls relevant, dass die Anpassungen innerhalb der existierenden App umgesetzt und keine eigenständige App für Tablets entwickelt werden soll. Die unter Android üblichen Installationsdateien – die APK-Dateien – sollen sowohl für Smartphone, als auch Tablet gleichermaßen gültig sein.

2.3 Anforderungen

Basierend auf den von Google monatlich veröffentlichen Statistiken zur Verteilung von Android und den aus Google Play bezogenen Daten zur Android-Verteilung für die App, lässt sich erkennen, dass die Android-Versionen 3.0 Honeycomb, 4.0 Ice Cream Sandwich, sowie 4.1 und 4.2 einen Gesamtmarktanteil von deutlich mehr als 50% ausmachen.

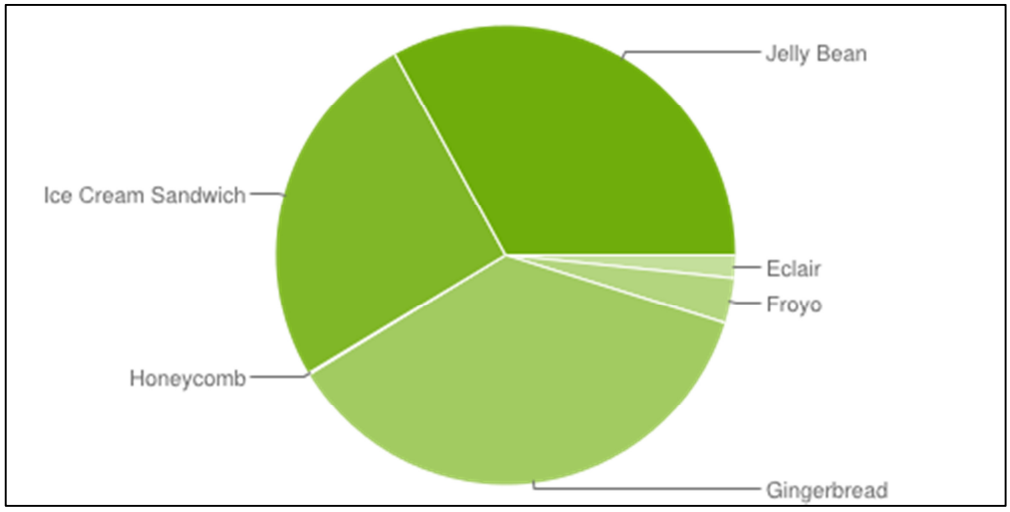

Abbildung 6 Android-Verteilung im Monat Juni 2013[10]

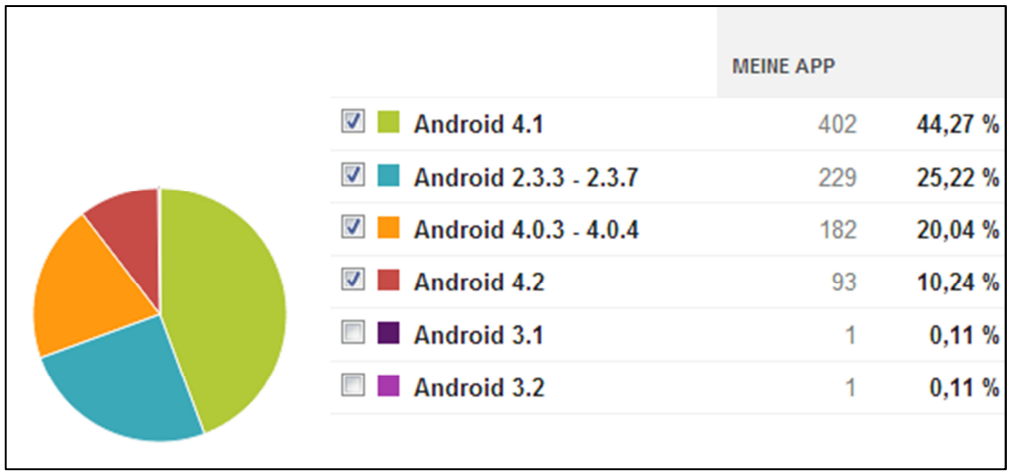

Abbildung 7 Verteilung von HSMWmobil im Monat Juni 2013[11]

[10] Vgl.: http://developer.android.com/about/dashboards/index.html, Juni 2013.

[11] Vgl.: https://play.google.com/apps/publish/?dev_acc=12464870337877015015&pli=1#StatsPlace:p=de.hsmw.app, Juni 2013.

Diese Aussage lässt sich sowohl auf den Gesamtmarkt von Android, als auch auf die App gleichermaßen anwenden.

Da für das Bachelor-Projekt nur jene Versionen relevant sind, die potentiell auf Tablet-Geräten lauffähig sind und von Google erst ab API Level 11 – Android 3.0 Honeycomb – Tablets offiziell unterstützt werden, müssen verschiedene Vorkehrungen getroffen werden, da die App ab API Level 10 – Android 2.3 Gingerbread – verfügbar ist.

Um den API Level für unterstützte Geräte nicht anheben zu müssen, wird, neben dem bereits genutzten Kompatibilitätspaket Android Support v4[12], auch die externe Support-Erweiterung ActionBarSherlock[13] integriert, die die namensgebende Action Bar auch für Geräte vor API Level 11 zur Verfügung stellt. Des Weiteren wurde bereits mit dem Update für Version 1.1 von HSMWmobil die Google Play Services-Bibliothek[14] integriert, welche unter anderem die Google Maps API v2 beinhalten.

[12] Kompatibilitätsbibliothek für Android, welches Funktionen und APIs aktueller Android-Version enthält.

[13] Vgl.: http://actionbarsherlock.com/index.html, 2013.

[14] Neustes Funktionspaket, welches aktuelle Funktionen und APIs von Google für Android enthält.

2.4 Projektmanagement

Auf Grund des Bearbeitungszeitraumes von 12 Wochen für die Bachelorarbeit, ist es notwendig eine durchdachte Projektplanung vorzunehmen, da diese ein wichtiger Eckpfeiler für den Erfolg eines jeden Projektes darstellt.

Durch die bemessene zeitliche Grenze ist dabei vor allem das Zeitmanagement von großer Bedeutung. Da Aufgaben nicht im Vorfeld exakt zeitlich abgegrenzt werden können, müssen entsprechende Pufferzeiten bereits eingeplant und festgelegt werden, um den Erfolg des Projektes nicht zu gefährden und zeitliche Probleme frühzeitig erkennen zu können.

Für eine bessere und effiziente Zeitplanung wird das Projekt in verschiedene Teilschritte zerlegt und Arbeitsschritte, sowie Meilensteine festgelegt.

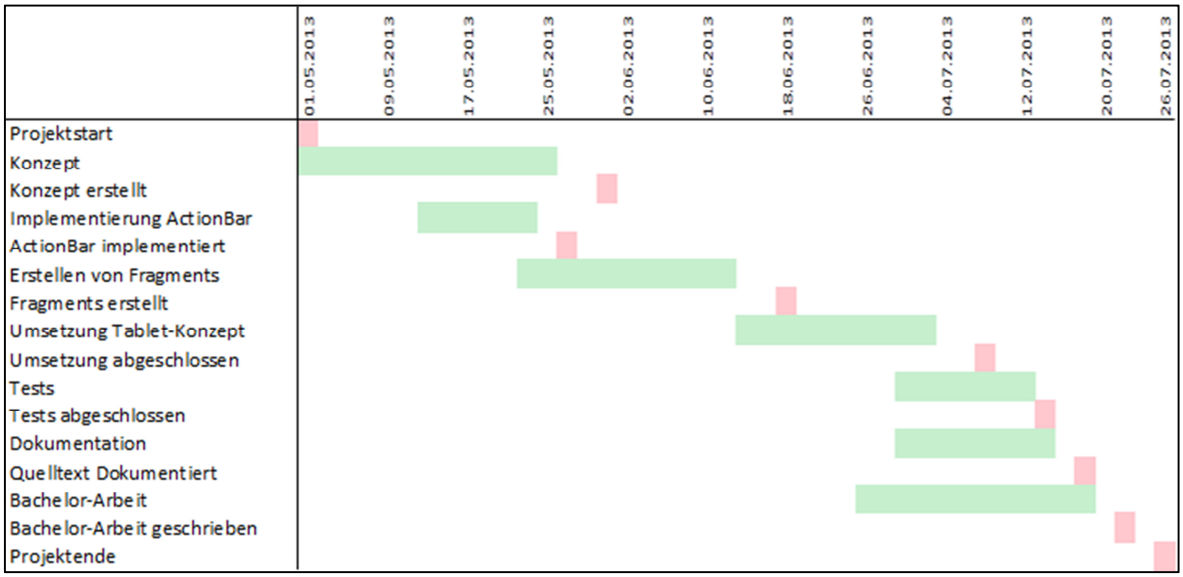

Abbildung 8 Zeitplanung für das Bachelor-Projekt

2.5 Entwicklungs- und Testumgebung

Als Entwicklungsumgebung für das Bachelor-Projekt, steht der Editor Eclipse, sowie das Android-SDK und die damit verbundenen Android Virtual Devices[15] für Debugging und Tests zur Verfügung. Dieser Editor eignet sich gut für die Entwicklung von Apps für Android, da von Google ein Repository für eine Eclipse-Erweiterung zur Programmierung bereitgestellt wird und viele Anleitungen in Büchern und Internet diese Entwicklungsumgebung vorschlagen und nutzen. Der Editor Eclipse liefert neben der Programmierfunktionalität außerdem die Möglichkeit, für verschiedene API-Versionen zu entwickeln, bietet einen Layout-Editor für die Entwicklung von Benutzeroberflächen, einen Echtzeittest auf einem Android-Emulator, einen Debugger für Fehlerlokalisation und -korrektur, und ermöglicht es die App direkt auf dem mobilen Endgerät auszuführen und zu testen. Die Google-eigene Entwicklungsumgebung Android Studio kann theoretisch ebenfalls für die Umsetzung verwendet werden, befindet sich allerdings noch in der Entwicklung und steht lediglich in einer frühen Vorabversion zur Verfügung. Da für die Arbeit während des Bachelor-Projektes jedoch eine stabile und ausgereifte Entwicklungsumgebung bevorzugt werden sollte und einige Funktionen noch nicht Einzug in Android Studio gehalten haben, wird Eclipse als Entwicklungsumgebung genutzt.

Durch die Verbindung zum Projekt „Campusapp“ kann auf die Ressourcen des Projektes zugegriffen werden, wodurch Testgeräte für die Entwicklung vorhanden und teilweise Neugeräte bestellt worden sind. Als Testgeräte stehen die Smartphones Samsung Galaxy Nexus, Samsuns Galaxy S Plus und Sony Ericsson X10 Mini zur Verfügung. Des Weiteren kann aus privatem Besitz auf die Testgeräte LG Nexus 4 und Sony Ericsson Arc S zurückgegriffen werden. Als Tablets sind das Asus Nexus 7, sowie das Sony Tablet S im Besitz der Hochschule vorhanden. Das private Tablet ViewPad 10s kann ebenfalls als Testgerät eingesetzt werden. Auf Grund der großen Anzahl an Testgeräten mit sowohl herstellerspezifischen, als auch Android-klassischen Oberflächen, kann ein breites Spektrum von Benutzerumgebungen getestet werden.

[15] Emulierte Testgeräte mit dem Betriebssystem Android.

3 Konzepte

3.1 Action Bar

Das Hauptnavigationselement unter Android, die Action Bar, welche mit Android 3.0 Honeycomb eingeführt wurde, befindet sich am oberen Bildschirm, ist Bestandteil einer jeden Activity und innerhalb der gesamten App sichtbar. Die Action Bar dient der Navigation innerhalb der App und repräsentiert den aktuellen Standort des Benutzers innerhalb der App.

Die Action Bar bietet dabei bereits standardmäßig verschiedene Funktionen, die dem Benutzer hilfreich sein können. So befindet sich auf der linken Seite der Action Bar das App-Icon der jeweiligen App, sowie rechts daneben der Titel der App. Dies dient der Übersichtlichkeit, da der Benutzer auf den ersten Blick darüber informiert ist, welche App gerade geöffnet ist. Des Weiteren befindet sich der Action Overflow in der Action Bar, welcher das bekannte Optionsmenü aus früheren Android-Versionen beinhaltet.

Abbildung 9 Action Bar mit App-Icon (1) und Action Overflow (2)

Alle Elemente des Optionsmenüs werden als Action Buttons bezeichnet und werden in zwei Kategorien unterschieden:

- FIT Action Buttons
- Less-frequently Action Buttons

Wichtige Elemente des Optionsmenüs können in der Action Bar außerhalb vom Action Overflow über Icons oder Texte innerhalb der Activity sichtbar gemacht werden, diese Elemente gehören zur Kategorie der FIT Action Buttons. FIT steht dabei für **F**requent (häufig) **I**mportant (wichtig) **T**ypical (typisch) und der Name bildet dabei die Faustregel für die Nutzung, wie sie seitens Google vorgesehen ist.[16]

[16] Vgl.: http://developer.android.com/guide/topics/ui/actionbar.html, 2013.

Die FIT Action Buttons sollen häufige, wichtige und typische Funktionen für den Benutzer schnell zugänglich machen. Werden diese Kriterien erfüllt, sollen diese Funktionen in der Action Bar Platz finden.

Die Gruppe der Less-frequently Action Buttons soll weniger häufig genutzte Funktionen im Action Overflow behalten, um die Übersichtlichkeit der App nicht zu gefährden und die Aktionen dennoch schnell zugänglich zu halten.

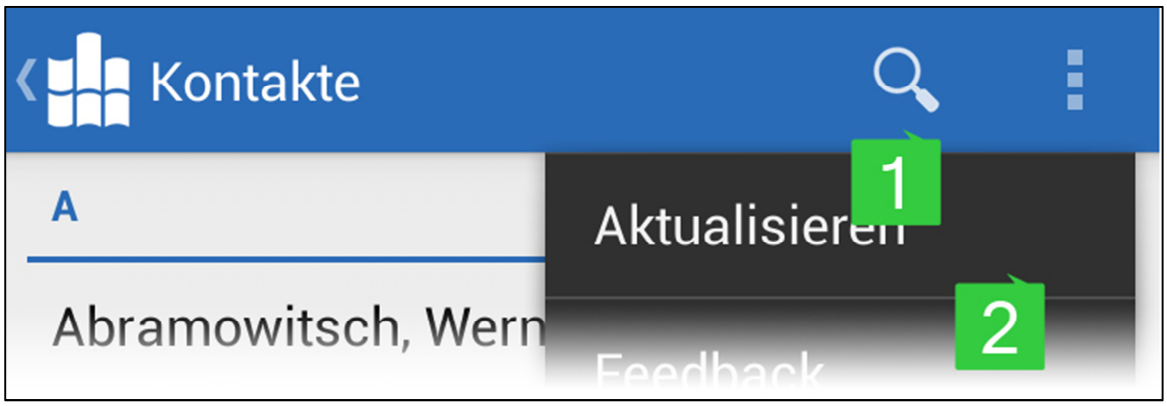

Abbildung 10 Action Bar mit Action Button (1) und Elementen des Action Overflow (2)

Es soll demnach genau überlegt werden, welche Funktionen für den Benutzer wichtig sind und wie die Produktivität und Übersichtlichkeit der Activities verbessert und optimiert werden kann.

Neben diesen grundlegenden Funktionen bietet die Action Bar jedoch noch weitere Funktionen und Bestandteile. So gibt es die Möglichkeit mit Hilfe von View Control verschiedene Views für die Anzeige von Inhalten zu nutzen und diese einfach zu wechseln. Dabei wird der Titel ausgeblendet und der View Control angezeigt. Wird die View Control geöffnet, wird ähnlich dem Action Overflow eine Liste mit möglichen Views eingeblendet und über eine entsprechende Auswahl die View gewechselt.

Abbildung 11 Action Bar mit View Control (1) und dessen Elementen (2)

Bei Nutzung der View Control ist es zwar nicht mehr möglich einen Titel zu vergeben, da dieser durch das gewählte Element automatisch geschieht, jedoch kann die View, welche von View Control eingeblendet wird, modifiziert werden. Beispielsweise durch Hinzufügen eines Untertitels (siehe Abbildung 11).

Ein weiteres Bedienkonzept, welches durch die Action Bar zur Verfügung steht ist die sogenannte Split Action Bar, welche die Action Bar in mehrere Teilbereiche zerlegt, um dem Benutzer somit mehr Interaktionsmöglichkeiten zur Verfügung zu stellen. Es können maximal drei verschiedene Teilbereiche genutzt werden:

- Main Action Bar
- Top Bar
- Bottom Bar

In jedem Teilbereich können verschiedene Elemente der Action Bar eingebettet werden, wobei lediglich die Main Action Bar fest zu vergeben ist. Die beiden anderen Teilbereiche sind jeweils optional und können ohne die Main Action Bar nicht existieren.

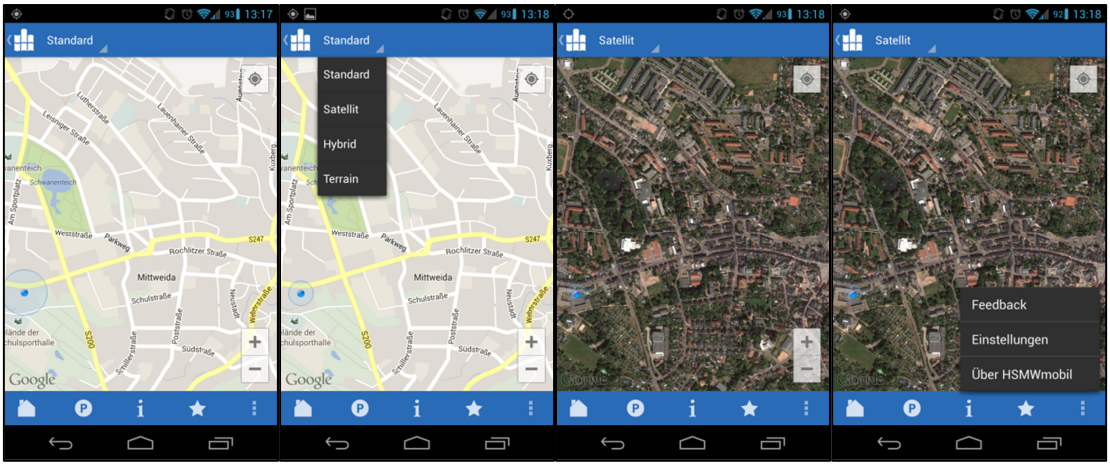

Abbildung 12 Nutzung von Bedienelementen der Action Bar im Campusplan

Im allgemeinen Fall wird bei Nutzung von Split Action Bar die Main Action Bar auf das Nötigste reduziert, sodass sich lediglich Navigationselemente oder der View Control darin befinden. Der Action Overflow, sowie die Action Buttons werden bei Nutzung der Bottom Bar in dieser platziert, so dass die gesamte Action Bar aufgeräumt und übersichtlich wirkt.

Die Top Bar soll dabei der Navigation dienen, so dass beispielsweise eine Tab-Ansicht genutzt und eingebettet werden kann. Die Tabs werden dabei im Portrait-Modus unterhalb der Main Action Bar zugänglich gemacht, wohingegen im Landscape-Modus die Tabs auf Grund des geringeren Platzangebotes in die Main Action Bar versetzt werden. Die Bottom Bar hingegen soll alle aktionsbedingten Elemente enthalten und ist diesen ausschließlich vorbehalten. Somit lassen sich Main Action Bar, Top Bar und Bottom Bar nach ihrer entsprechenden Funktion klassifizieren, so dass ein sinnvolles Konzept für jede App existiert.

Die Action Bar soll als Navigationselement genutzt werden, was durch die View Control bereits verdeutlicht wird. Jedoch existiert in der Action Bar noch ein weiteres Element, welches der Navigation dient. Dieses Element findet links neben dem App-Icon Platz und soll eingeblendet werden, sobald die aktuelle Ansicht nicht der Startansicht – der Main-Activity – der App entspricht. Dieses Element dient der Zurück-Navigation innerhalb der App.

Abbildung 13 Action Bar App-Icon mit Zurück-Navigation (1)

Im Gegensatz zur Zurück-Taste, welche zu dem zuletzt aufgeführten Task aus dem Back Stack[17] zurück navigiert, dient diese Navigation dazu in der App-Hierarchie eine Ebene nach oben zu gelangen. So ist es möglich, von einer gleichrangigen Activity eine neue Activity zu öffnen und von dieser, über die Navigation der Action Bar, in der Hierarchie nach oben zu navigieren. Dahingehend wird über die Zurück-Taste zu der vorherigen Activity im Back Stack navigiert. An oberster Stelle ist dabei die Startansicht, welche keine weitere hierarchische Navigation ermöglicht.

[17] Ansammlung alle aufgerufenen Activities in zeitlicher Anordnung.

Ein weiteres und wichtiges Bedienkonzept, welches mit der Action Bar eingeführt wurde, stellt eine eingebettete Suchleiste dar, die sogenannte Search View[18] als Action View[19]. Dabei wird als Action Button eine Such-Aktion zur Verfügung gestellt und bei Betätigung werden der Titel und die Action Buttons in der Action Bar ausgeblendet, sowie eine Suchleiste an deren Stelle eingeblendet. Der Inhalt der dargestellten Suchleiste kann mit Hilfe eines Listeners[20] ausgelesen und die entsprechenden gefilterten Daten innerhalb der View der Activity oder des Fragments dargestellt werden.

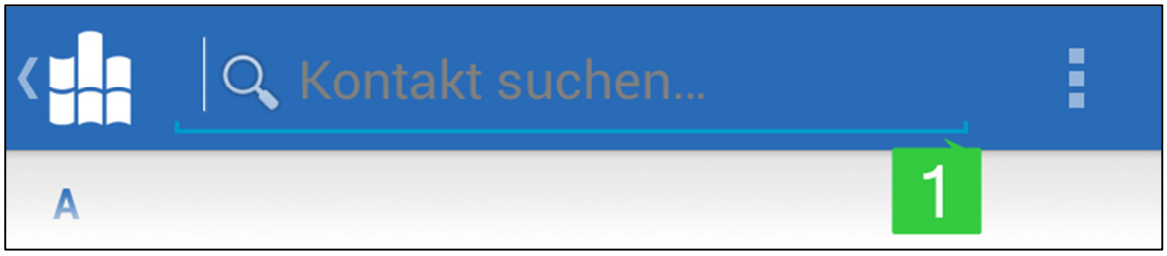

Abbildung 14 Action Bar mit Search View (1)

Für die Action Bar existieren noch weitere, für die Bachelorarbeit jedoch nicht relevante Möglichkeiten zur sinnvollen Nutzung. So stellt die Action Bar ebenfalls einen Action Provider[21] zur Verfügung, der eine Art Liste einblendet, welche weitere Aktionen beinhalten kann. Des Weiteren ist es möglich eine Contextual Action Bar[22] zu nutzen, die Kontext-basiert auf Interaktionen des Benutzers, wie das Auswählen von Text, reagiert und entsprechende Aktionen, wie beispielsweise das Kopieren, zur Verfügung stellen kann.

Zukünftig ist nicht zu erwarten, dass Google das Konzept der Action Bar verwerfen wird und ein neues Konzept ausarbeitet, dass an die Stelle der Action Bar tritt. Es ist eher davon auszugehen, dass dieses Konzept mit steigendem API Level mehr und mehr Funktionen erhält, welche die Navigation und den Funktionsumfang erweitern.

---

[18] Element einer Action View[19].

[19] Werkzeug welches als Ersatz bei Betätigung eines Action Button erscheint.

[20] Werkzeug um Eingabe-Ereignisse zu behandeln.

[21] Stellt ein Menü mit Items für die Navigation zur Verfügung.

[22] Konzept einer Kontext-bezogene Action Bar.

3.2 Activities und Fragments

Die vorherrschende Komponente unter Android ist nach wie vor die Activity, welche Interaktionen von Benutzern mit der App über eine View ermöglicht. Dabei existieren verschiedene Möglichkeiten die View nach dem eigenen Belieben zu gestalten und zu erschaffen. Innerhalb einer App existiert immer genau eine Main-Activity, welche im Manifest festgelegt ist und beim Starten der App geöffnet wird. Es ist jedoch möglich einem Projekt weitere Activities zuzuordnen und zwischen den einzelnen Activities zu navigieren. Jede Activity stellt dabei ihren eigenen Inhalt zur Verfügung und zeigt diesen an. Dieses Konzept existiert seit den Anfängen von Android und hat sich grundlegend kaum verändert. Auch mit API Level 17 bleibt die Activity die grundlegende Komponente von Android.

Zu den mit API Level 11 eingeführten Neuerungen gehören die sogenannten Fragments, welche ähnlich der Activity mit dem Benutzer interagieren und Inhalte darstellen können. Fragments unterscheiden sich jedoch in einem Punkt maßgeblich von Activities: Fragments können ohne eine Activity nicht existieren. Das heißt Activities können Fragments beinhalten, müssen dies jedoch nicht aber ein Fragment muss immer zu einer Activity hinzugefügt werden. Dabei werden die Fragments zu einer sogenannten View Group[23] innerhalb einer Activity hinzugefügt und sie besitzen einen eigenen Lebenszyklus – oder auch Lifecycle genannt. Sie sind aber dennoch vom Lebenszyklus der Activity abhängig. Mit Hilfe von Fragment Transaction[24] können Fragments zu einer View Group hinzugefügt, ersetzt oder entfernt werden. Fragments können im Gegensatz zu Activities auch mit anderen Fragments zusammengefügt werden und somit eine zusammengehörige Darstellungsform bilden. Des Weiteren können Fragments auch ohne Benutzeroberfläche existieren, wohingegen eine Activity immer mit einer Benutzeroberfläche verknüpft ist.

Fragments haben weitere positive Eigenschaften, die sie von einer Activity unterscheiden. So ist es möglich Fragments an verschiedenen Stellen wiederzuverwenden, was mit einer Activity nur bedingt möglich ist. Auf Grund dieser wichtigen Eigenschaften ist es für das Bachelor-Projekt nötig, die Benutzeroberfläche mit Hilfe von Fragments neu zu gestalten.

[23] Gruppe zusammengehöriger Views.

[24] Stellt Operationen für die Verwaltung von Fragments zur Verfügung.

3.2.1 Layouts

Mit Hilfe von Fragments lassen sich – wie in **3.2** bereits erwähnt – neue Konzepte für Design-Entscheidungen erstellen. Fragments lassen sich innerhalb einer View Group zusammenfügen und bilden somit eine eigene Darstellungsform. Dieses Verhalten kann dazu verwendet werden, sich vom klassischen Konzept der Activities unter Android zu verabschieden, dass für jede Ansicht eine eigene Activity vorsieht.

Abbildung 15 Klassisches Konzept der Activities unter Android

Der neue Ansatz sieht jedoch vor, die Inhalte in Form von Fragments innerhalb einer Activity darzustellen. Dazu wird der Inhalt mit Hilfe von Fragment Transaction ausgetauscht, so dass das aufrufende Fragment im Back Stack untergebracht wird, während das aufgerufene Fragment im Vordergrund erscheint und für Interaktionen zur Verfügung steht. Das Wechseln der Fragments übernimmt dabei die Activity, so dass eine Callback-Methode[25] eingebaut werden muss, welche die Activity darüber informiert, die Fragment Transaction durchzuführen und den Datenaustausch zwischen den Fragments erledigt.

Abbildung 16 Einsatz von Fragment Transaction

[25] Rückruffunktion in der Softwareentwicklung

Die Zurück-Navigation erfolgt bei diesem Konzept über den Back Stack von Android, welcher es über die Activity ermöglicht Fragment Transactions rückgängig zu machen und den vorherigen Stand einer Ansicht wiederherzustellen. Dabei wird beim Aufruf von Fragment Transaction das aufrufende Fragment angehalten, mit Hilfe der Activity in den Back Stack verschoben und beim Betätigen der Zurück-Taste wieder aufgerufen.

Für Tablet-Bildschirme gibt es zudem die Möglichkeit ein separates Layout zu definieren und somit zwei Fragments nebeneinander darzustellen. Das neue Layout beinhaltet anschließend beide Fragments und platziert diese entsprechend in der Activity. Dieses Konzept bietet sich daher für Listen mit einer Detailansicht an. Bei Auswahl eines Listenelements wird die Detailansicht lediglich aktualisiert und nicht, wie beim Öffnen von einer Activity, neu erstellt. Des Weiteren wird keine neue Activity aufgerufen, so dass die Aktualisierung direkt zur Laufzeit in der aktuellen Activity stattfindet.

Abbildung 17 Nebeneinander platzierte Fragments auf einem Tablet

Zu diesem Zweck sollen Fragments stets wiederverwendbar gestaltet und programmiert werden. Die Fragments können dabei für verschiedene Layout-Konfigurationen genutzt werden, um somit den zur Verfügung stehenden Platz je nach Gerät und Platzangebot optimal ausnutzen zu können.

Neben der Action Bar und dem Activity-Fragment-Prinzip, gibt es noch ein weiteres wichtiges Konzept, welches für die Entwicklung von Apps für Tablets essentiell wichtig ist: Flexible Layouts. Diese Form der Layouts wird erstellt, da nicht nur eine Form von Smartphones oder Tablets existiert, sondern in verschiedenen Größen und mit verschiedenen Seitenverhältnissen auf dem Markt vorhanden sind. Flexible Layouts werden mit Hilfe von relativen Maßen erstellt, zu diesem Zweck gibt es unter Android die Einheit dp[26]. Diese Einheit berechnet in Abhängigkeit von der physikalischen Pixeldichte, die relative Größe im Vergleich zu 160dpi.

[26] Density-independent Pixels – Von der Auflösung unabhängige Einheit.

4 Implementierung

4.1 Grundlagen

Vor Beginn der Umsetzung muss der aktuelle Quelltext aus dem SVN[27] der Hochschule Mittweida heruntergeladen. Um unter Eclipse mit der freien Versionsverwaltung arbeiten zu können, muss ein Plug-In in Eclipse integriert werden. Die Wahl des Plug-Ins ist dabei dem Entwickler überlassen, da verschiedene Plug-Ins zur Verfügung stehen. In diesem Projekt wird jedoch Subversive[28] verwendet, welches die grundlegenden Funktionen für Update, Commit und Synchronisation zur Verfügung stellt.

Um nicht am produktiven Projekt zu arbeiten, wird eine lokale Kopie des SVN-Arbeitsprojektes angelegt, da sonst nicht parallel am Quelltext der aktuellen, in Google Play verfügbaren Version gearbeitet werden kann, um Fehler schnellstmöglich zu beseitigen. Bei Bedarf kann dieser sogenannte Fork[29] im SVN unter einem neuen Projektnamen geführt und mit diesem aktuell gehalten werden. Im späteren Verlauf der Entwicklung werden die Neuerungen und Änderungen aus dem Originalprojekt in den Fork übernommen, sowie der Fork als Originalprojekt klassifiziert.

Des Weiteren wird die aktuelle Android Kompatibilitätsbibliothek Support Library v4 heruntergeladen und dem Projekt hinzugefügt. Diese Bibliothek stellt sicher, dass einige Funktionen von Android, welche mit einem höheren API Level eingeführt wurden, auch unter dem benötigtem API Level 10 zur Verfügung stehen.

Im letzten Schritt wird die Bibliothek Google Play Services SDK heruntergeladen und in Eclipse importiert. Diese Bibliothek stellt beispielsweise die neue Google Maps API v2 zur Verfügung, welche es ermöglicht ein Map-Fragment in das Projekt zu integrieren oder die neuen Möglichkeiten der Google Maps-API zu nutzen. Da Google Play Services als Projekt für Eclipse vorliegt, muss das Projekt von Hand mit Hilfe der Projekteinstellungen als Bibliothek hinzugefügt werden. Des Weiteren wird das Build Target[30] von API Level 10 auf API Level 17 geändert, um unter aktuellen Geräten das native Design verwenden und beispielsweise die Action Bar nativ nutzen zu können. Durch die Änderung des Build Targets werden veraltete Methoden hervorgehoben, so dass eine Überarbeitung von Teilen des Quelltextes zu empfehlen ist.

---

[27] Eine freie Versionsverwaltung.

[28] Vgl.: http://www.eclipse.org/subversive/, 2013.

[29] Eine Abspaltung eines Software-Projektes.

[30] API-Version, welche für das Kompilieren des Projektes verwendet werden soll.

4.2 Action Bar

4.2.1 Integration

Das Konzept der Action Bar wurde bereits unter **4.1** erläutert und soll an dieser Stelle umgesetzt werden. Da jedoch API Level 10 als Minimalanforderung existiert und die Action Bar erst mit API Level 11 zur Verfügung steht, muss auf eine externe Bibliothek zurückgegriffen werden, die die Action Bar auch für ältere API Level zur Verfügung stellt. Zu diesem Zweck eignet sich ActionBarSherlock, da die Bibliothek als Android-Projekt zur freien Verfügung steht. Es muss jedoch innerhalb der App erwähnt werden, dass ActionBarSherlock verwendet wird.

Das Projekt muss aus dem Downloadbereich der Homepage[31] von ActionBarSherlock heruntergeladen und als Projekt in Eclipse importiert werden. Um die Bibliothek für das eigene Projekt verwenden zu können, muss, wie bereits bei der Bibliothek Google Play Services, das Projekt mit Hilfe der Projekteinstellungen als Bibliothek hinzugefügt werden. Eclipse bietet beim Hinzufügen nur die geeigneten Projekte an, so dass ActionBarSherlock schnell gefunden werden sollte.

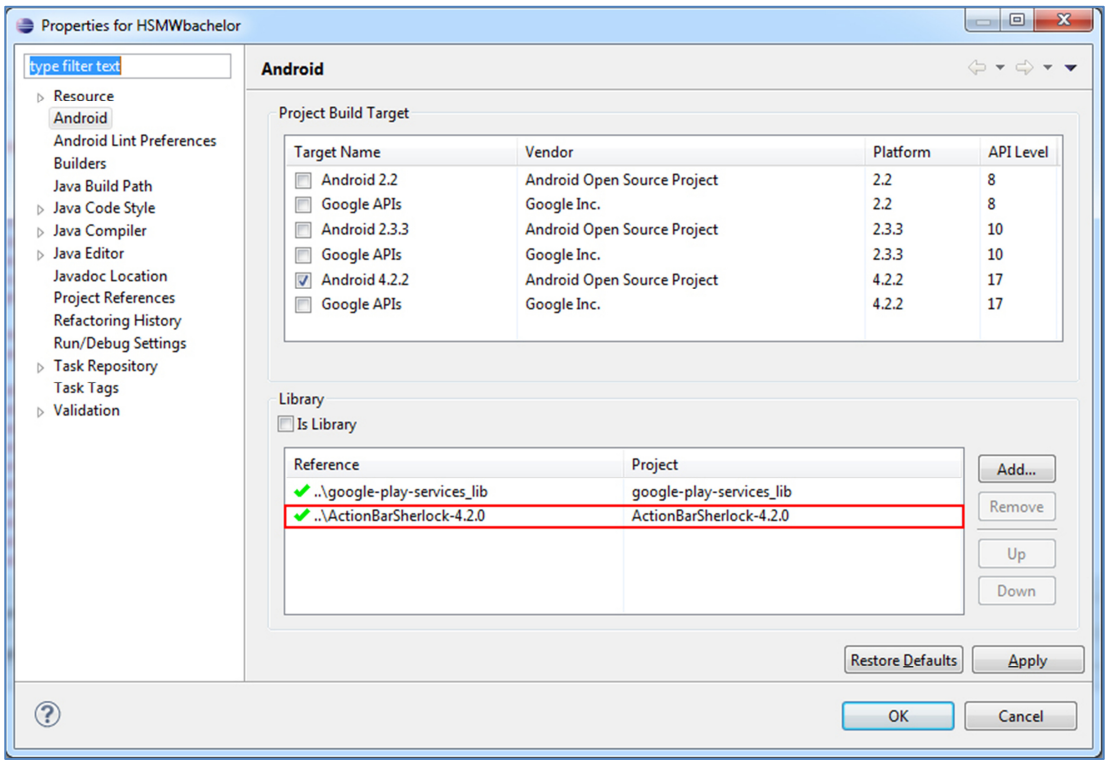

Abbildung 18 Projekteinstellungen unter Eclipse für Android-Projekte

[31] Vgl.: http://actionbarsherlock.com/download.html, 2013.

Nach dem erfolgreichen Hinzufügen der Bibliotheken kann die Action Bar integriert werden. Um auf die Action Bar innerhalb des Quelltextes zugreifen zu können, wird an Stelle der Methode *getActionBar()* die Methode *getSupportActionBar()* verwendet.

4.2.2 Theme

Zum Zeitpunkt der Entwicklung von HSMWmobil in Version 1.0 stand auf Grund des niedrigen API Level die Action Bar nicht zur Verfügung. Vor API Level 11 existierte eine sogenannte Title Bar, welche ein App-Icon, sowie den Titel der App beinhaltete. Um diese auszublenden, wurde mit Hilfe der Datei *styles.xml*, welche das Design einer App beschreibt, die Title Bar ausgeblendet, um mehr Inhalt darstellen zu können.

Mit dem Anstieg des API Level und mit Hilfe von ActionBarSherlock ist die Action Bar jedoch auch unter API Level 10, welchen die App HSMWmobil als Mindestvoraussetzung besitzt, verfügbar. Das Design der Action Bar richtet sich an das definierte Theme, welches im Manifest hinterlegt ist. Da die Hochschule ein eigenes Corporate Design besitzt und dieses innerhalb der App Anwendung finden soll, muss ein eigenes Theme erstellt werden, welches neben der Action Bar auch alle weiteren Elemente einer App nach Belieben verändern und beschreiben kann. Als Grundlage für das Theme wird *Light.DarkActionBar* der externen Bibliothek ActionBarSherlock gewählt. Um das Theme entsprechend den Farben des Corporate Design der Hochschule zu gestalten, muss die Datei *styles.xml* bearbeitet und folgende Attribute gesetzt werden:

```
<style name="Widget.HSMW.ActionBar" parent="Widget.Sherlock.ActionBar">
   <item name="background">@color/blue_hsmw</item>
   <item name="android:background">@color/blue_hsmw</item>
   <item name="backgroundSplit">@color/blue_hsmw</item>
   <item name="android:backgroundSplit">@color/blue_hsmw</item>
   <item name="titleTextStyle">@style/Widget.HSMW.ActionBar.TitleTextStyle</item>
   <item name="android:titleTextStyle">@style/Widget.HSMW.ActionBar.TitleTextStyle</item>
   <item name="subtitleTextStyle">@style/Widget.HSMW.ActionBar.SubTitleTextStyle</item>
   <item name="android:subtitleTextStyle">@style/Widget.HSMW.ActionBar.SubTitleTextStyle</item>
   <item name="android:icon">@drawable/ic_launcher_actionbar</item>
   <item name="icon">@drawable/ic_launcher_actionbar</item>
</style>
<style name="Widget.HSMW.ActionBar.TitleTextStyle"
   parent="TextAppearance.Sherlock.Widget.ActionBar.Title">
   <item name="android:textColor">@color/white</item>
</style>
<style name="Widget.HSMW.ActionBar.SubTitleTextStyle"
   parent="TextAppearance.Sherlock.Widget.ActionBar.Subtitle">
   <item name="android:textColor">?android:attr/textColorSecondaryInverse</item>
</style>
```

Abbildung 19 Auszug aus der Datei *styles.xml*

Das erstellte Theme erhält den Namen *Theme.HSMW* und verändert lediglich die Action Bar, sowie einige annehmbare Formen von dieser. Das Theme wird anschließend innerhalb des Manifests des Projektes definiert:

```
<application
    android:name=".HsmwApplication"
    android:allowBackup="false"
    android:icon="@drawable/ic_launcher_hsmw"
    android:label="@string/app_name"
    android:theme="@style/Theme.HSMW" >

    [...]

</application>
```

Abbildung 20 Auszug aus dem Manifest

Die Action Bar wird nun innerhalb der App global angezeigt, es sind jedoch keinerlei Funktionen implementiert, so dass die Action Bar noch nicht funktional gestaltet ist. Um diesen Umstand zu ändern, wird die Navigation in den einzelnen Modulen umgesetzt. Dazu wird innerhalb der Activity auf die Action Bar zugegriffen und mit Hilfe der Methode *setDisplayHomeAsUpEnabled()* und dem Wert *true* als Parameter die Navigation aktiviert.

```
ActionBar actionbar = getSupportActionBar();
actionbar.setDisplayHomeAsUpEnabled(true);
```

Abbildung 21 Navigation innerhalb der Action Bar aktivieren

Um in Fragments auf die Action Bar zugreifen zu können, muss mit Hilfe des Kontextes die Methode *getSupportActionBar()* aufgerufen werden. Die Methode *setDisplayHomeAsUpEnabled()* aktiviert das Navigationselement, implementiert dieses jedoch nicht. Dies muss innerhalb von *onOptionsItemSelected(MenuItem item)* erledigt werden. Dazu wird ein neuer Case[32] erstellt, der bestimmt, was geschehen soll, wenn das Navigationselement betätigt wird. Diese Navigation wird für die einzelnen Module separat übernommen.

[32] Eintretender Fall einer Fallunterscheidung.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
    case android.R.id.home:
        Intent intent = new Intent(getActivity(), HsmwMainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(intent);
        getActivity().finish();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }

}
```

Abbildung 22 *onOptionsItemSelected(MenuItem item)* mit Intent zur Navigation

4.2.3 Action Buttons und Action Overflow

Im nächsten Schritt muss genauestens überlegt werden, welche Items als Action Button implementiert werden sollen und welche Items im Action Overflow unterzubringen sind. Für die Beurteilung wird die Regelung[33] für Action Buttons herangezogen, die besagt, dass Action Buttons häufig (**F**requent), wichtig (**I**mportant), typisch (**T**ypical) sein sollen, um Anwendung zu finden. Als solche Items eignen sich innerhalb von HSMWmobil nur wenige Elemente, da viele Aktionen innerhalb der View erledigt werden und kaum wiederkehrende, wichtige Funktionen innerhalb der Module existieren. Einzig das Aktualisieren innerhalb der Module News, Blog, Medien-Mittweida, Stundenplan, Notenanzeige und Mensaplan eignet sich als Action Button.

Die Umsetzung erfolgt ebenfalls als Case innerhalb der Methode *onOptionsItemSelected(MenuItem item)*. Für die Aktualisierung wird ein neuer Download-Thread gestartet, welcher das XML-Dokument entsprechend der Module herunterlädt, parst und als Vektor an die jeweilige Activity oder das entsprechende Fragment weiterleitet. Die Activity oder das Fragment aktualisiert daraufhin mit Hilfe eines Handlers die Benutzeroberfläche.

Im Action Overflow werden überwiegend die gleichen Items untergebracht. Impressum, Einstellungen und Feedback sollen innerhalb der gesamten App aufrufbar sein. Um nicht in jeder Activity die entsprechenden Intents aufrufen zu müssen und somit Redundanz zu erzeugen, wird für diesen Zweck das Prinzip der Vererbung genutzt.

[33] Vgl.: http://developer.android.com/guide/topics/ui/actionbar.html, 2013.

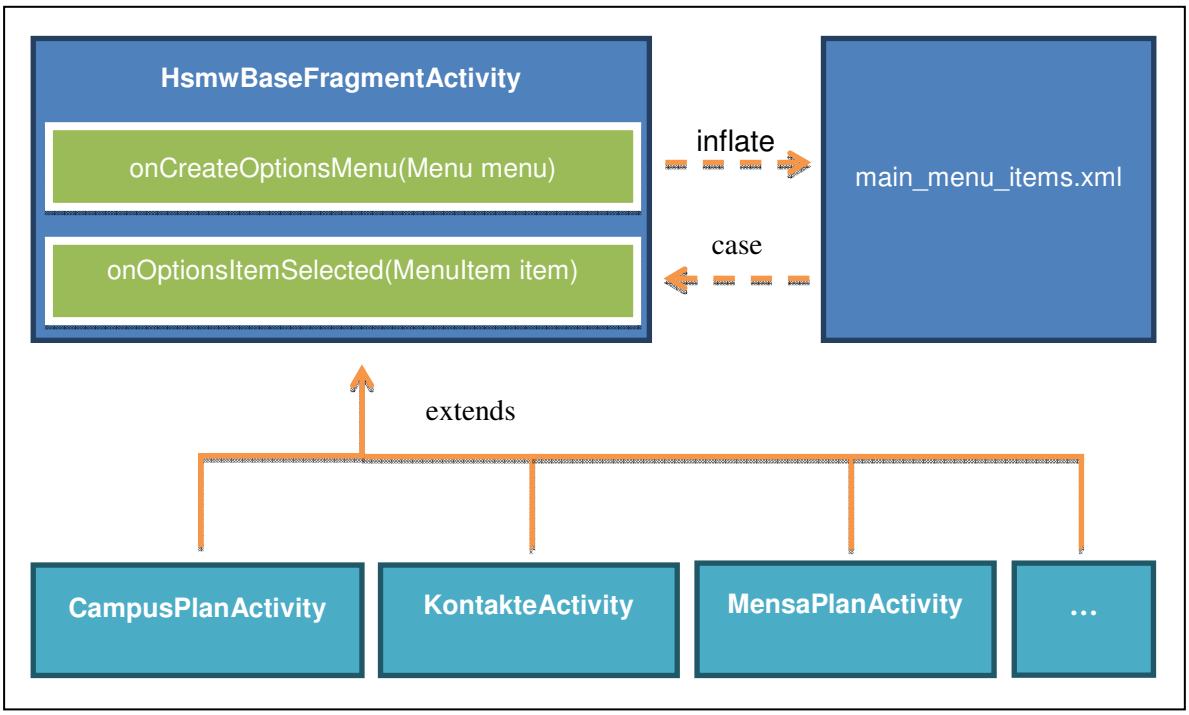

Abbildung 23 Funktionsprinzip von *HsmwBaseFragmentActivity*

Dazu wird eine neue Klasse *HsmwBaseFragmentActivity* erzeugt, welche *SherlockFragmentActivity* erweitert und die Methoden *onCreateOptionsMenu(Menu menu)*, sowie *onOptionsItemSelected(MenuItem item)* überschreibt. Innerhalb dieser Methoden wird das Menü *main_menu_items.xml* initialisiert und den Items eine Funktion verliehen, indem jedem Item ein entsprechender Case zugeordnet und innerhalb des eintretenden Falls ein entsprechender Intent gestartet wird.

Diese Klasse wird anschließend von den Activities innerhalb des Projektes erweitert, so dass die Items für Impressum, Einstellungen und Feedback des Action Overflows von überall erreichbar sind.

Weitere Items im Action Overflow, wie die Anmeldung bzw. Abmeldung, sowie die modulspezifischen Items, werden äquivalent zur *HsmwBaseFragmentActivity* innerhalb der entsprechenden Activity implementiert.

4.2.4 View Control

Als weitere Funktion der Action Bar für HSMWmobil soll die Control View integriert werden, die eine Änderung der Darstellung mit Hilfe verschiedener Elemente, speziell eines Drop Down-Feldes, ermöglicht, um Inhalte auf unterschiedliche Art und Weise darstellen zu können. Diese Funktion wird innerhalb der Module Campusplan, Stundenplan und Notenanzeige implementiert. Im Campusplan soll die View Control genutzt werden, um zwischen den verschiedenen Kartentypen *Normal, Satellite, Hybrid* und *Terrain* wechseln zu können. Dabei wird die Map-View aktualisiert und mit Hilfe der Methode *setMapType()* der Kartentyp festgelegt. Dazu wird die Action Bar wie folgt aufgerufen:

```
ActionBar actionBar = mActivity.getSupportActionBar();
actionBar.setDisplayHomeAsUpEnabled(true);
actionBar.setDisplayShowTitleEnabled(false);
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
```

Abbildung 24 Aufruf der Action Bar mit View Control

Die View Control selbst benötigt einen Adapter, welcher alle Wahlelemente enthält und bestimmt, was passieren soll, wenn ein Element ausgewählt wird. Die Elemente werden mit Hilfe eines Arrays, welches die Namen enthält, über den Adapter zur View hinzugefügt. Dem Adapter muss dabei der Kontext übergeben werden, welcher mit Hilfe der Methode *getThemedContext()* abgefragt wird. Zusätzlich wird mit Hilfe von ActionBarSherlock das Layout für die Drop-Down-Liste festgelegt. Im letzten Schritt wird der Activity mit Hilfe der Methode *setListNavigationsCallbacks()* der Adapter hinzugefügt, sowie ein neuer Listener instanziiert. Je nach Auswahl wird ein entsprechender Case aufgerufen, in dem der Kartentyp mit Hilfe der Methode *setMapType()* verändert wird.

```
ArrayAdapter<CharSequence> spinnerAdapter = new ArrayAdapter<CharSequence>(
                actionBar.getThemedContext(),
                com.actionbarsherlock.R.layout.sherlock_spinner_dropdown_item, items);
actionBar.setListNavigationsCallbacks(spinnerAdapter, new OnNavigationListener() {
        public boolean onNavigationItemSelected(int itemPosition, long itemId) {
                mMapTypeIndex = itemPosition;
                if (mMap == null) return true;
                        (mMapTypeIndex) {
                                case 0: mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
                                        break;
                                case 1: mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
                                        break;
                                case 2: mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
                                        break;
                                case 3: mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
                                        break;
                        }
                return true;
        }

});
```

Abbildung 25 Array Adapter und Callback des View Controls

Ebenfalls Einzug in den Campusplan soll die Split Action Bar halten, welche es ermöglicht mehr Elemente als Action Buttons zu verwenden. Dazu wird die Action Bar innerhalb des Campusplans in die Main Action Bar und die Bottom Bar aufgeteilt. Um dies zu bewerkstelligen muss die Activity im Manifest folgendermaßen definiert werden:

```
<activity
    android:name="de.hsmw.app.campusplan.CampusPlanActivity"
    android:label="@string/campusplan"
    android:uiOptions="splitActionBarWhenNarrow" >

</activity>
```

Abbildung 26 Einstellung der Split Action Bar im Campusplan

Damit die Action Buttons innerhalb der Split Action Bar angezeigt werden können, muss den Items innerhalb der Menü-XML das Attribut *android:showAsAction* auf den Wert *ifRoom* geändert werden. Wie der Name vermuten lässt, wird das Item lediglich als Action Button angezeigt, wenn genügend Platz innerhalb der Action Bar zur Verfügung steht. Ansonsten befindet sich das Item im Action Overflow.

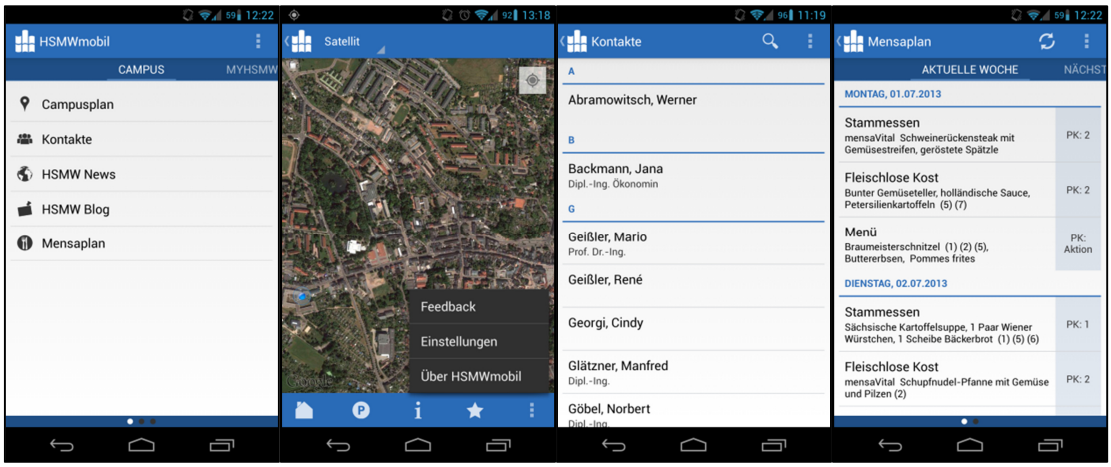

Abbildung 27 Einsatz von Action Bar innerhalb von HSMWmobil

Integriert werden die Items mit Hilfe der Methoden *onCreateOptionsMenu(Menu menu)* und *onItemSelected(MenuItem item)* wird die Funktionalität der Items festgelegt. Auch an dieser Stelle funktioniert dies äquivalent der Integration innerhalb der Hilfsklasse *HsmwBaseFragmentActivity*.

Neben dem Campusplan soll die Control View in die Module Stundenplan, sowie Notenanzeige integriert werden. Dazu wird, wie bereits im Campusplan, die Action Bar initialisiert, sowie mit Hilfe der Methode *setNavigationMode()* und dem gewünschten Modus, welcher über den *int*-Parameter *actionBar.NAVIGATION_MODE_LIST* bestimmt wird, eine Drop-Down-Liste realisiert. Des Weiteren muss mit Hilfe eines instanziierten Adapters, welcher die Integration der Menü-XML, sowie die Darstellung der View Control übernimmt, ein entsprechender Navigations-Callback implementiert werden. Über die Items des Navigations-Callbacks können die Views ausgewählt werden.

Für den Adapter wird eine weitere Hilfsklasse *ActionBarTwoLineDropDownNavigationAdapter* integriert, welche einen Adapter zur Verfügung stellt, der eine Drop-Down-Liste, die zwei Zeilen innerhalb der Liste enthält, abbildet. Der Adapter erweitert *ArrayAdapter<CharSequence>* und beinhaltet neben einem Konstruktor, welcher beim Aufruf den Kontext und ein Array mit den Ansichten als Parameter voraussetzt, die Methode *getView()*, welche wiederum *getView()* aus der vererbenden Klasse *ArrayAdapter* überschreibt. Diese Methode liefert eine View zurück, die mit Hilfe des Adapters eingeblendet wird. Das Layout wird innerhalb der Layout-Datei *actionbar_twoline_spinner_item* festgelegt, die aus einem Linear Layout und den zwei enthaltenen Text Views besteht. In der View werden die zwei Text Views instanziiert, die Titel und Untertitel für die zwei Zeilen der Liste darstellen. Des Weiteren wird in der Methode mit Hilfe von *setText()* der Titel und der Untertitel gesetzt. Der Titel entspricht der jeweiligen Ansicht und wird aus dem Array, welcher als Parameter durch den Konstruktor zur Verfügung steht, ausgelesen. Der Untertitel wird über die Hilfsklasse H*smwApplication*, welche bereits seit Version 1.0 von HSMWmobil integriert ist und grundlegende Funktionen der An- und Abmeldung enthält, abgefragt und der Anmeldenamen eingeblendet.

Im Stundenplan kann dadurch entweder zwischen einer Tagesansicht, welche ein Blättern mit Hilfe eines View Pagers zur Verfügung stellt oder einer Wochenansicht, eine einfache List View, gewählt werden. Für die Änderung der Ansicht existieren die Methoden *showViewPager()* und *showListView()*, welche durch Auswahl der Items aufgerufen werden. Der Aufruf der Methoden erfolgt innerhalb der Methode *onNavigationItemSelected(int itemPosition, long itemId)*, die an den Navigations-Callback gebunden ist.

In der Notenanzeige existiert die Methode *setUpDropDownNavigation()*, die die Action Bar initialisiert, den Navigation Type festlegt und mit Hilfe eines Adapters, äquivalent zum Stundenplan, eine View Control implementiert. Für die Übergabe des Parameters wird für das Array ein XML-Dokument, welches die Seminargruppen des Benutzers enthält, aus dem Internet aufgerufen und geparst. Die View Control dient innerhalb der Notenanzeige dazu, zwischen den einzelnen Seminargruppen zu wählen. Entsprechend der Auswahl der Seminargruppe innerhalb der View Control, wird mit Hilfe der Methode *onNavigationItemSelected(int itemPosition, long itemId)* der zugehörige Download-Thread gestartet und eine Liste der eigenen Noten in Form einer List View angezeigt. Eine Auswahl der Noten öffnet abschließend die entsprechende Detailansicht.

4.2.5 Action View – Search View

Action Views sind umfangreicher zu implementieren und anfällig für Fehler. Eine spezielle Form von Action View – die Search View – soll innerhalb des Moduls Kontakte genutzt werden. Für die Umsetzung bedarf es eines Action Buttons, welcher für die Suche zur Verfügung steht. Dieser wird mit Hilfe der Menü-XML als zusätzliches Item implementiert:

```
<item
    android:id="@+id/contacts_search"
    android:actionLayout="@layout/contacts_search_text"
    android:icon="@drawable/ic_search"
    android:orderInCategory="0"
    android:showAsAction="always|collapseActionView"
    android:actionViewClass="com.actionbarsherlock.widget.Search View"
    android:title="@string/search" />
```

Abbildung 28 Item für die Search View im Modul Kontakte

Das Item muss für die Nutzung als Action View spezielle Attribute, neben den Standard-Attributen für ID, Icon und Ordnungsnummer für Items der Action View, enthalten. Dem Item muss mitgeteilt werden, dass es als sogenannte Search View fungieren soll. Dazu muss das Attribut *android:actionViewClass* mit dem Wert *com.actionbarsherlock.widget.*SearchView gesetzt werden. Auf Grund des API Level von HSMWmobil muss auf die Bibliothek ActionBarSherlock an Stelle der nativen Action Bar zugegriffen werden. Damit sich die View öffnen lässt, muss diese also aufklappbare View gekennzeichnet und immer angezeigt werden.

Zu diesem Zweck wird das Attribut *android:showAsAction* mit dem Wert *always|collapseActionView* verwendet. Da die View eine Text View für die Suche integrieren soll, muss ein Layout erstellt werden, welches eine Text View enthält. Dieses Layout wird als *contacts_search_text* erzeugt und innerhalb des Items mit Hilfe des Attributs *actionLayout* und einer Verlinkung zur Layout-Datei implementiert.

Die weitere Implementierung erfolgt in der Methode *onCreateOptionsMenu(Menu menu)*, welche die Menü-XML erweitern muss. Innerhalb der Methode wird ein neuer Search Manager instanziiert, der auf den Kontext der Action Bar zugreifen und den Search Service auslesen muss. Des Weiteren muss eine Search View instanziiert werden, welche über die ID auf die Action View zugreifen kann.

```
SearchManager searchManager = (SearchManager)activity.getSystemService(Context.SEARCH_SERVICE);
Search View Search View = Search View)menu.findItem(R.id.contacts_search).getActionView();
Search View.setSearchableInfo(searchManager.getSearchableInfo(activity.getComponentName()));
```

Abbildung 29 Instanziierung von Search Manager und Search View

Als letzter Schritt wird der Search View ein *OnQueryTextListener* hinzugefügt, der verschiedene Ereignisse steuern kann, wenn etwas mit der Search View geschieht. Diese Ereignisse können das Abschicken eines Suchbegriffs – *onQueryTextSubmit()* – oder die Eingabe von Text innerhalb der Search View – *onQueryTextChanged()* – sein. Für die Suche innerhalb der Kontakte soll eine Echtzeitsuche verwendet werden, welche die List View der Kontaktliste während der Eingabe von Text in die Search View aktualisiert, um die Ergebnisse schnell ausliefern und dem Benutzer zur Verfügung stellen zu können.

Für die Suche wird ein Vektor mit den Kontaktelementen erzeugt, der später die gefilterten Suchergebnisse enthalten soll. Dieser Vektor wird zu Beginn des Suchalgorithmus geleert, da es sich nicht um eine feste, einmalige Suche handelt und sonst die gefilterten Elemente am Ende des Vektors hinzugefügt werden würden. Als weitere Variable für die Suche wird die Länge des eingegebenen Textes in der Search View benötigt. Da die Methode *onQueryTextChanged()* aufgerufen wird, sobald sich der Text innerhalb der Search View ändert, entspricht die Textlänge immer exakt der Eingabelänge.

Für das Füllen des Vektors wird eine Schleife verwendet, die den Eingabetext der Search View mit den Elementen des Ursprungsvektors *listenElemente* vergleicht. Der Vergleich findet dabei auf zwei Ebenen statt – dem Nach- und dem Vornamen.

```
if (Pattern.compile(Pattern.quote(newText.toString()),
                Pattern.CASE_INSENSITIVE).matcher(listenElemente.get(i).getVorname()).find()) {
        suchElemente.add(listenElemente.get(i));
} else if (Pattern.compile(Pattern.quote(newText.toString()),
                Pattern.CASE_INSENSITIVE).matcher(listenElemente.get(i).getName()).find()) {
        suchElemente.add(listenElemente.get(i));

}
```

Abbildung 30 Ausschnitt des Suchalgorithmus

Die Suche nach passenden Einträgen erfolgt innerhalb der Suche Case-insensitiv, was bedeutet, dass die Groß- und Kleinschreibung für die Suche keine Rolle spielt. Wird ein passender Eintrag gefunden, der dem Eingabetext der Search View entspricht, wird dieser Eintrag dem neuen Vektor hinzugefügt. Anschließend wird mit Hilfe des Adapters die List View aktualisiert.

4.3 Umstellung auf Fragments

Die Umstellung auf Fragments ist eine Fleißaufgabe und erfordert lediglich für die Fehleranalyse erweiterte Kenntnisse der Android-Programmierung. Für die Umsetzung muss strikt nach Vorgabe gearbeitet werden, damit am Ende der Umstellung eine strukturierte Programmierung, die einen Zielparameter darstellt, vorhanden ist.

Im ersten Schritt wird die Activity des Campusplans umgestellt, welche eine Map-View enthält. Dabei ist die wesentliche Aufgabe aus der Map-View ein Map-Fragment zu erstellen. Die Map-View wurde innerhalb der Activity initialisiert, instanziiert und implementiert. Diese Aufgabe übernimmt die *getView()*-Methode der neu erstellten Klasse *CampusPlanMapFragment*. Die Klasse erweitert *SupportMapFragment*, welche durch die Kompatibilitätsbibliothek von Google zur Verfügung steht. Innerhalb von *getView()* wird eine View instanziiert, die das Layout des Campusplans erweitert. Mit Hilfe der View und einer Referenz auf die Activity, die durch die Methode *getActivity()* und dem Cast *SherlockFragmentActivity* innerhalb des Fragments initialisiert wird, wird ein Zugriff auf alle Views des Layouts sichergestellt.

Während der Umstellung müssen Methoden, welche im Fragment nicht direkt aufrufbar sind, aber für die Darstellung der Inhalte oder die Implementierung der Funktionen benötigt werden, mit Hilfe des Kontextes, der View oder der Activity, aufgerufen werden. Beispielsweise wird die Methode *findViewById()* innerhalb des Fragments mit der Referenz auf die View aufgerufen.

```
view.findViewById(R.id.cp_base_layout);
```

Abbildung 31 View referenzieren

Das Referenzieren von Elementen über die Activity, das Fragment, den Kontext oder die anderen Views, erfolgt dabei äquivalent zur Vorgehensweise zum Referenzieren einer View.

Das Erstellen des Fragments ist dabei nur ein Teil, der erledigt werden muss. Zusätzlich muss ein Container als Layout erstellt werden, dem die Fragments zur Laufzeit der Activity hinzugefügt werden können. Der Container ist ein einfaches Frame Layout mit der ID *campusplan_fragment_container*. Innerhalb der Activity *CampusPlanActivity* wird in der Methode *onCreate()* überprüft, ob der Container leer ist. Wenn der Container leer ist, wird das Map-Fragment *CampusPlanMapFragment* instanziiert und mit Hilfe einer Fragment Transaction zum Container hinzugefügt.

```
getSupportFragmentManager().beginTransaction().add(
        R.id.campusplan_fragment_container, fragment).commit();
```

Abbildung 32 Fragment Transaction

Dieses Konzept wird im Zuge der Umstellung auf alle weiteren Activities und Module übertragen, da es für die Tablet-Optimierung notwendig ist. Dabei wird je Modul ein Container erstellt, dem die Fragments über die Activity per Fragment Transaction hinzugefügt werden. In diesem Zuge wird der Quelltext optimiert und bereinigt, sowie eine gewisse Struktur, welche in allen Klassen weitestgehend gleich vorhanden sein soll, umgesetzt.

Der Stundenplan stellt bei der Umstellung auf Fragments eine Besonderheit dar, da diese zwei verschiedene Listen für die Darstellung des Stundenplans enthält. Neben der klassischen List View, welche die Stunden untereinander in einer Liste mit den jeweiligen Datumsangaben platziert, existiert noch eine Sortierung der Stunden nach Tag innerhalb eines View Pagers. Im View Pager selbst werden für die einzelnen Tage jeweils Fragments aufgerufen, durch die, mit Hilfe einer Wischgeste, gewechselt werden kann. Jedes Fragment enthält dabei einen eigenen List Adapter.

4.4 Zwei-Spalten-Layout

Um das zusätzliche Platzangebot auf größeren Displays, wie beispielsweise Tablets nutzen zu können, wird das Prinzip des Zwei-Spalten-Layouts an sinnvollen Stellen umgesetzt. Dabei werden die zuvor erstellten Fragments nebeneinander platziert. Über Callbacks ist es möglich ein Fragment aus dem jeweilig anderen heraus zu aktualisieren oder zu modifizieren. Genutzt werden soll dieses Verhalten innerhalb von HSMWmobil, um Listen- und Detailansichten zu realisieren. Dabei stellt die Listenansicht und die Detailansicht jeweils ein eigenes Fragment dar. Am Beispiel des Moduls Kontakte wird im folgenden Abschnitt das Prinzip und die Umsetzung selbst erläutert.

Wird ein Element aus der Liste ausgewählt, wird die Position des Listenelements an die Activity übertragen. Die Activity aktualisiert anschließend die Detailansicht anhand der Position des Listenelements.

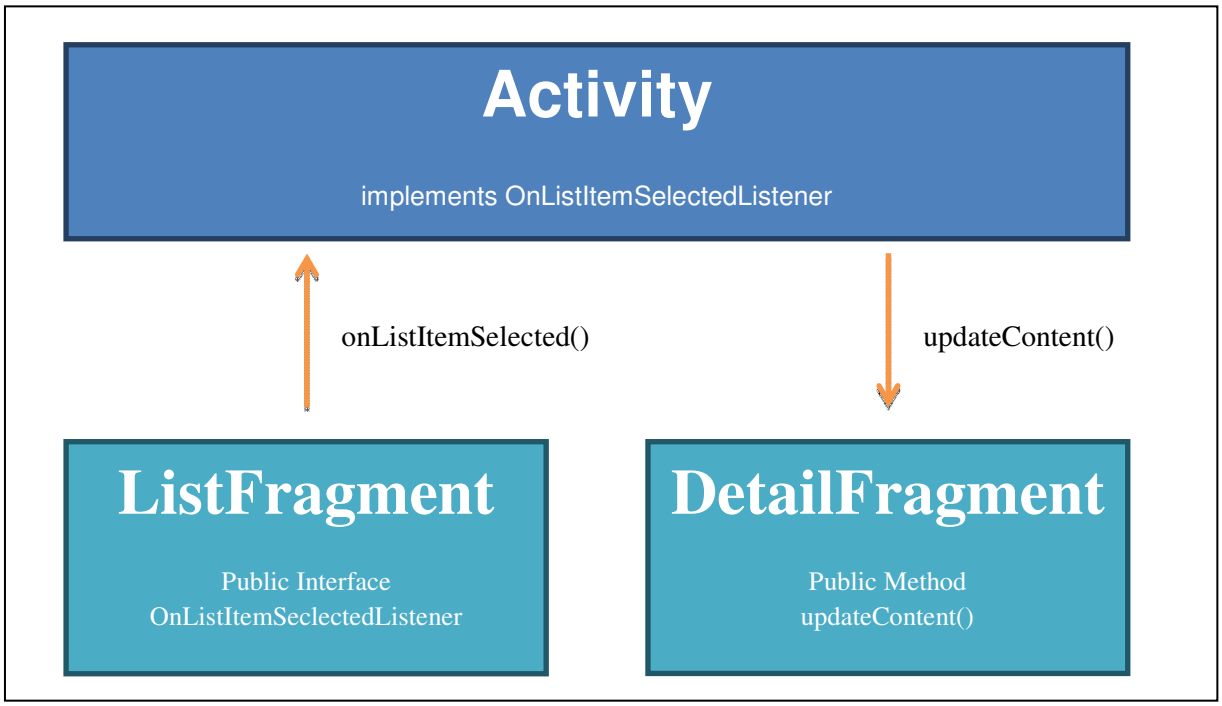

Abbildung 33 Konzept für die Umsetzung des Zwei-Spalten-Layouts

Für die Umsetzung werden zwei verschiedene Layouts benötigt, wobei eine für normale Displayauflösungen aufgerufen wird und die andere bei großen Displays Anwendung findet. Die Version des Layouts für Smartphones wird innerhalb des Layout-Ordners abgelegt und beinhaltet ein Frame Layout, welches später als Container für die Fragments dienen soll.

```
<?xml version="1.0" encoding="utf-8"?>
<Frame Layoutt xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/kontakte_fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Abbildung 34 Fragment Container in HSMWmobil

Das Layout für Tablets wird innerhalb des Ordners *layout-large* abgelegt, der speziell für große Displays vorhanden ist. Innerhalb des Layouts wird ein Linear Layout hinzugefügt, dass das Attribut *android:orientation* mit dem Wert *horizontal* enthält. Dieses Layout ruft die beiden Fragments direkt auf und platziert diese in Folge des Attributs *android:orientation* nebeneinander. Durch das Attribut *android:layout_weight* wird die Gewichtung der einzelnen Fragments festgelegt. Grundlegend wird an dieser Stelle festgelegt, dass das Fragment, welches die List View enthält eine Gewichtung mit dem Wert *1*, und das Fragment für die Detailansicht eine Gewichtung mit dem Wert *2* erhalten soll. Dementsprechend werden die Werte für das Attribut *android:layout_weight* innerhalb der einzelnen Fragments festgelegt.

```
<?xml version="1.0" encoding="utf-8"?>
<Linear Layout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false" >

    <fragment android:name="de.hsmw.app.kontakte.KontaktListFragment"
              android:id="@+id/kontakt_list_fragment"
              android:layout_weight="1"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

    <fragment android:name="de.hsmw.app.kontakte.KontaktDetailFragment"
              android:id="@+id/kontakt_detail_fragment"
              android:layout_weight="2"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

</Linear Layout>
```

Abbildung 35 Layout für ein Zwei-Spalten-Layout

Nachdem die Layouts festgelegt sind, muss innerhalb der Activity auf die entsprechende Displaygröße reagiert werden. Da auf Smartphones lediglich der Container für die Fragments implementiert, jedoch kein Fragment selbst integriert ist, muss in diesem Fall ein Fragment für die Listenansicht zu dem Container hinzugefügt werden. Zu diesem Zweck wird mit Hilfe einer *if*-Abfrage geprüft, ob das Fragment bereits vorhanden ist. Ist kein Fragment vorhanden wird eine neue Instanz des Fragments instanziiert und mit Hilfe von Fragment Transaction zum Fragment Container hinzugefügt. Auf Tablets geschieht an dieser Stelle nichts, da kein Fragment Container vorhanden ist.

Nachdem die Layouts definiert sind, ist die Benutzeroberfläche vorhanden aber innerhalb der Fragments geschieht nichts, da die Fragments zwar existieren, aber nicht miteinander verbunden sind – die Fragments interagieren nicht miteinander.

Um diesen Zustand zu ändern, muss ein Interface für die Activity geschaffen werden, welches dem Fragment die Möglichkeit bietet über einen sogenannten Callback Informationen an die Activity zu senden. Ein neues Interface wird dabei mit Hilfe eines Zugriffsmodifikators – für ein Interface in den meisten Fällen *public* – und dem Rückgabetypen deklariert. Die beinhaltende Methode des Interface, ist die Methode, die das Interface aufruft und welche in der implementierenden Klasse des Interfaces zur Verfügung stehen muss. Diese Methode würde von Eclipse automatisch als „*unimplemented method*“ angezeigt und hinzugefügt, wenn die Methode nicht im Quelltext vorhanden ist.

```
public interface OnContactSelectedListener {
        public void onContactSelected(int position, int UID);

}
```

Abbildung 36 Interface für die Kommunikation von Activity und Fragment

Um sicherzustellen, dass die aufrufende Activity das Interface implementiert hat, muss eine neue Instanz des Listeners instanziiert werden, die mit Hilfe der Activity umgewandelt wird. Um zu verhindern, dass die App an dieser Stelle mit einer *NullPointerException* abbricht, weil kein Interface innerhalb der Activity implementiert ist, wird der Cast des Listeners innerhalb eines *try-catch*-Blocks durchgeführt.

```
try {
        mActivityCallback = (OnContactSelectedListener)activity;
} catch (ClassCastException e) {
        throw new ClassCastException(context + " must implement OnContactSelectedListener");

}
```

Abbildung 37 Generierung einer *ClassCastException* bei fehlendem Interface

Im *catch*-Block wird im Falle, dass eine *ClassCastException* generiert wird, der Exception[34] die Information des fehlenden Interfaces übergeben. Die App würde in diesem Fall abstürzen und die *ClassCastException* mit Hinweis auf das fehlende Interface liefern. Diese Information kann mit Hilfe von LogCat in Eclipse aufgerufen werden, sollte das Interface nicht von der Activity implementiert sein.

```
FATAL EXCEPTION: main
java.lang.ClassCastException:   de.hsmw.app.kontakte.KontaktActivity must implement
                                OnContactSelectedListener
```

Abbildung 38 Generierte Exception bei fehlendem Interface

Welche Daten zwischen Fragment und Activity ausgetauscht werden, ist dabei frei wählbar. Es ist möglich nach Belieben Daten zwischen Fragment und Activity auszutauschen. Dabei ist es ebenso unerheblich, welcher Datentyp ausgetauscht werden soll.

Beispielsweise wird im Modul Kontakte neben der aktuellen Position des Elements der List View auch die UID des gewählten Kontakts übertragen. Die Position wird innerhalb der Activity dazu verwendet, den gewählten Listeneintrag zu markieren. Die UID wird vom Detail-Fragment benötigt, da weitere Daten heruntergeladen werden müssen. Mit Hilfe der UID können speziell für den gewählten Kontakt die Daten innerhalb des Detail-Fragments per XML abgefragt, heruntergeladen, geparst und dargestellt werden.

[34] Ausnahme die durch einen Fehler auftritt und mit Hilfe einer Ausnahmebehandlung verarbeitet werden kann.

Im Modul News wird ebenfalls die Position des Listeneintrages übermittelt. Da die Detailansicht eine Web View enthält, die den entsprechenden Newseintrag als Internet-Seite aufruft, wird lediglich die URL benötigt. Als Design-Entscheidung wurde beschlossen, dass die Action Bar dazu genutzt werden soll, die aktuelle Überschrift des Newseintrages und das Datum der Veröffentlichung darzustellen. Damit dies realisiert werden kann, müssen die Informationen ebenfalls an das Detail-Fragment übertragen werden. Um die Daten gesammelt und effizienter zu übertragen, werden alle Informationen einem Bundle hinzugefügt, das anschließend mit Hilfe des Interfaces an die Activity übertragen wird. Die Activity aktualisiert abschließend das Detail-Fragment anhand der Daten.

Innerhalb der Notenanzeige wird ebenfalls die Position des Listeneintrages übermittelt. Zusätzlich werden die Seminargruppe und die Modulinformationen übertragen. Diese werden jeweils als String-Parameter an das Fragment übergeben.

Im Stundenplan wird, wie in den anderen Module auch, die Position übertragen. Des Weiteren wird die jeweilige Stunde als Objekt *StundenPlanStunde* an die Activity übermittelt. Das Interface wird dabei äquivalent zu dem Interface des Moduls Kontakte implementiert. Bis auf den Namen ändert sich am Prinzip der Kommunikation über Interfaces nichts.

Um Daten an das Interface übergeben zu können, muss das Interface innerhalb des Fragments aufgerufen werden und die Daten als Parameter übergeben werden. Dabei werden die Daten über den Callback an die Activity übertragen, indem die Activity als Referenz die Methode selbst aufruft. Im Falle der Module Kontakte, News, Notenanzeige und Stundenplan wird, entgegen der üblichen Implementierung innerhalb des Fragments, das Interface innerhalb des List Adapters implementiert. Diesem Vorgehen liegt zu Grunde, dass das On-Click-Event innerhalb des List Adapters implementiert worden ist und somit die Daten vom List Adapter an die Activity geliefert werden müssen.

```
public void onClick(View view) {
        mActivityCallback.onContactSelected(pos, kontaktElement.getUid());
        view.getFocusables(pos);
        view.setSelected(true);
}
```

Abbildung 39 Aufruf eines Callbacks

Die Methoden *getFocusables(pos)* und *setSelected(true)* auf die View regeln dabei das Markieren des gewählten Items innerhalb der List View, um dieses hervorzuheben. Die Methode *getFocusables(pos)* liefert alle fokusierbaren Views innerhalb der View. Als ausgewähltes Element wird das Listenelement mit Hilfe der Methode *setSelected(true)* markiert und greift dabei auf die gelieferten Ergebnisse von *getFousables(pos)* zurück. Zusätzlich muss die List View den Wert *CHOICE_MOD_SINGLE* als Attribut erhalten.

Im nächsten Schritt muss das Interface innerhalb der Activity aufgerufen werden, um die Datenverarbeitung zu integrieren. Die Activity muss damit das Interface – im Falle des Moduls Kontakte – *OnContactSelectedListener* implementieren. Innerhalb von Eclipse sollte der Debugger reagieren und einen Fehler generieren, der besagt, dass fehlende Methoden vorhanden sind, welche implementiert werden müssen. Um den Fehler zu beheben, muss die Methode *onContactSelected(int position, int uid)* hinzugefügt werden.

```
public class KontaktActivity extends HsmwBaseFragmentActivity
        implements OnContactSelectedListener {
        @Override
        public void onContactSelected(int position, int UID) {
                // TODO Auto-generated method stub


        }
}
```

Abbildung 40 Klassen- und Methoden-Rumpf des Interfaces

Da innerhalb von *onContactSelected(int position, int uid)* das Detail-Fragment auf Tablets aktualisiert beziehungsweise auf Smartphones hinzuzufügt werden soll, müssen entsprechende Vorkehrungen innerhalb des Details-Fragments getroffen werden. Aus diesem Grund muss zum einen der Aufruf und zum anderen die Übergabe der Parameter angepasst werden, da diese bisher über die Argumente des Fragments implementiert sind.

Dazu wird eine neue Methode geschaffen, welche die Aktualisierung des Inhalts übernehmen soll und aus der Activity heraus aufgerufen werden kann. Dieser Methode werden die Parameter der Activity Position und UID übergeben.

```
public void updateDetailView(int position, int uid) {

}
```

Abbildung 41 Methodenrumpf für das Aktualisieren des Inhalts

Das Detail-Activity wird in diesem Zuge aufgeräumt und strukturiert, um einzelne Komponenten aktualisieren zu können. Innerhalb der *updateDetailView(int position, int uid)* werden die Methoden *setActionBar()*, *setFaculty()*, *setAddress()*, *setPhones()*, *setFax()*, *setMails()*, *setTimes()* und *setContactVisibility()* aufgerufen, die jeweils einen Teil der Benutzeroberfläche aktualisieren oder die Anzeige, welcher Inhalt in der View angezeigt werden soll, übernehmen. Diese Methoden sind spezifisch und unterscheiden sich in den anderen Modulen.

Für die weitere Umsetzung werden mehrere statische *String*-Variablen benötigt. Der Lifecycle des Fragments ist zwar abhängig vom Lifecycle der Activity, jedoch besitzt jedes Fragment einen eigenen Lifecycle, weswegen dieser von jedem Fragment selbst verwaltet werden muss. Diese statischen *String*-Variablen werden unter anderem für die Speicherung des Zustandes der App durch *SavedInstanceState()* benötigt, um diese eindeutig zuordnen zu können.

```
public final static String ARG_POSITION = "position";
public final static String ARG_UID = "uid";
public final static String ARG_PHONE_INDEX = "phone_index";
public final static String ARG_ELEMENT = "element";
public final static String ARG_BITMAP = "bitmap";
```

Abbildung 42 Statische *String*-Variablen für die Speicherung des Zustandes

Um den Status des Fragments zu sichern, wird die Methode *onSaveInstanceState()* überschrieben, die alle für den Inhalt relevanten Variablen zwischenspeichert und welche beim Wiederaufruf beziehungsweise beim Wiederherstellen des Fragments benötigt werden. Innerhalb der Methode wird dem Bundle *outState* die jeweiligen Variablen für Position, UID, den Index-Wert für Telefonnummern, das Kontaktelement selbst und das Bitmap hinzugefügt.

```
@Override
public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt(ARG_POSITION, mCurrentPosition);
        outState.putInt(ARG_UID, mCurrentUid);
        outState.putInt(ARG_PHONE_INDEX, mPhoneIndex);
        outState.putParcelable(ARG_ELEMENT, mDetailElement);
        outState.putParcelable(ARG_BITMAP, mBitmap);

}
```

Abbildung 43 Methode *onSaveInstanceState()* innerhalb des Details-Fragments

Um den Status des Fragments wiederherstellen zu können, wird innerhalb von *onCreateView()* überprüft, ob das Bundle *savedInstanceState* leer ist. Wird das Fragment wiederaufgerufen beziehungsweise wiederhergestellt, ist das Bundle mit den jeweiligen Objekten belegt und diese können abgerufen und wiederhergestellt werden. Der Inhalt des Bundle wird anschließend auf die entsprechenden Variablen verteilt, welche innerhalb von *onCreateView()* oder anderen Methode benötigt werden.

```
if (savedInstanceState != null) {
        mCurrentPosition = savedInstanceState.getInt(ARG_POSITION);
        mCurrentUid = savedInstanceState.getInt(ARG_UID);
        mPhoneIndex = savedInstanceState.getInt(ARG_PHONE_INDEX);
        mDetailElement = savedInstanceState.getParcelable(ARG_ELEMENT);
        mBitmap = savedInstanceState.getParcelable(ARG_BITMAP);

}
```

Abbildung 44 Wiederherstellung der Objekte aus *savedInstanceState()*

Der Lifecycle des Detail-Fragments ist somit abgeschlossen und die Umsetzung des Zwei-Spalten-Layouts für Tablets kann fortgesetzt werden. Wird die Activity oder das Fragment von der Dalvik-Maschine bereinigt, kann mit Hilfe von *savedInstanceState* die Activity oder das Fragment wiederhergestellt werden.

Die Information für Position und UID sollen mit Hilfe von Argumenten beim Aufruf des Detail-Fragments übergeben werden beziehungsweise die Methode *updateDetailView(int position, int uid)* des Detail-Fragments direkt aufgerufen werden, wenn das Fragment bereits vorhanden ist. Um innerhalb des Detail-Fragments die Verarbeitung zu starten, wird die Methode *onStart()* der Activity genutzt, welche immer aufgerufen wird, wenn das Fragment erzeugt oder längere Zeit gestoppt und wiederaufgerufen wird.

Innerhalb der Methode *onStart()* wird ein neues Bundle instanziiert und mit Hilfe der Methode *getArguments()* wird das Bundle beim Aufruf des Fragments aus der Activity übernommen. Nachdem das Bundle instanziiert wurde, muss eine *if*-Abfrage gestartet werden, die überprüft ob das Bundle leer – also *null* – ist. Im Falle eines leeren Bundles wird die Methode *updateDetailView(int position, int uid)* mit den Parametern für Position und UID aus der Activity aufgerufen. Dazu wird auf das Bundle *args* die Methode *getInt()* jeweilig aufgerufen. Als Parameter der Methode *getInt()* werden die statischen Strings *ARG_POSITION* und *ARG_UID* verwendet.

```
@Override
public void onStart() {
        super.onStart();
        Bundle args = getArguments();
        if (args != null) {
            updateDetailView(args.getInt(ARG_POSITION), args.getInt(ARG_UID));
        } else if (mCurrentPosition != -1 && mCurrentUid != 0) {
            updateDetailView(mCurrentPosition, mCurrentUid);

        }

}
```

Abbildung 45 Methodenaufruf von *onStart()* zum Aktualisieren des Fragments

Ist das Bundle *args* leer, soll überprüft werden, ob die Standard-Werte für Position und UID Bestand haben. Ist das nicht der Fall, wird die Methode *updateDetailView(int position, int uid)* mit den Parametern, welche zuvor aus dem Bundle *savedInstaceState* gelesen wurden, aufgerufen. Ist das Bundle *args* leer und haben die Standard-Werte noch Bestand, wird an dieser Stelle nichts ausgeführt und das Fragment wird zwar aufgerufen, jedoch wird der Inhalt nicht aktualisiert. Lediglich ein leeres Fragment wird in diesem Fall angezeigt. Da dieser Zustand nur auf Tablets auftreten kann, wird anschließend ein *else*-Zweig und ein Layout hinzugefügt. Das Layout beinhaltet ein Text View, welches eine Information für die Auswahl eines Kontaktes enthält. Im *else*-Zweig wird mit Hilfe der Methode *setVisibility()* die Sichtbarkeit der Layouts so festgelegt, dass der Hinweis zur Auswahl eines Kontaktes sichtbar und die anderen Layouts den Wert *GONE* erhalten, der dafür sorgt, dass diese Layouts nicht sichtbar sind. Des Weiteren muss die Methode *setContactVisibility()* so erweitert werden, dass der Hinweis beim Aktualisieren des Inhalts durch *updateDetailView(int position, int uid)* ausgeblendet wird.

Als letzter Schritt muss der Aufruf für die Aktualisierung beziehungsweise das Hinzufügen des Fragments aus der Activity heraus implementiert werden. Dazu muss die Methode *onContactSelected(int position, int uid)* angepasst werden. Über einen Fragment Manger und die Methode *findFragmentById()* wird das Detail Fragment initialisiert, sofern dieses vorhanden ist.

```
KontaktDetailFragment mDetailFragment = (KontaktDetailFragment)
        getSupportFragmentManager().findFragmentById(R.id.kontakt_detail_fragment);
```

Abbildung 46 Initialisieren des Detail Fragments innerhalb der Activity

Das initialisierte Fragment wird mit Hilfe einer *if*-Abfrage auf den Wert *null* überprüft, um feststellen zu können, ob das Detail-Fragment innerhalb der View existiert. Das Detail-Fragment existiert dabei lediglich auf Tablets beziehungsweise Geräten, welche den Ordner *layout-large* nutzen. Ist das Fragment vorhanden und initialisiert, wird mit Hilfe des Fragments die Methode *updateDetailView(int position, int uid)* aufgerufen. Als Parameter für Position und UID werden die Objekte *position* und *uid*, die mit Hilfe des Interfaces an die Activity übertragen wurden, verwendet.

```
if (mDetailFragment != null) {
        mDetailFragment.updateDetailView(position, uid);
} else { }
```

Abbildung 47 Aktualisieren des Inhalts innerhalb der Activity

Ist das Fragment nicht vorhanden, wird ein neues Detail-Fragment, sowie ein neues Bundle instanziiert und dem Bundle die *int*-Werte von Position und UID hinzugefügt. Des Weiteren wird dem Fragment mit Hilfe der Methode *setArguments()* das Bundle definiert. Innerhalb des Detail-Fragments wird auf das Bundle, wie bereits erwähnt, zugegriffen und die Daten verarbeitet.

```
KontaktDetailFragment newFragment = new KontaktDetailFragment();
Bundle args = new Bundle();
args.putInt(KontaktDetailFragment.ARG_POSITION, position);
args.putInt(KontaktDetailFragment.ARG_UID, uid);
newFragment.setArguments(args);
```

Abbildung 48 Hinzufügen eines Bundles mit Hilfe von Argumenten

Im letzten Schritt innerhalb der Activity muss eine Fragment Transaction vollzogen werden. Dazu wird eine neue Fragment Transaction instanziiert und mit Hilfe der Methoden *replace()* und *commit()* das Fragment ersetzt, sowie die Fragment Transaction durchgeführt.

```
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
transaction.replace(R.id.main_fragment_container, newFragment);
transaction.commit();
```

Abbildung 49 Neue Fragment Transaction

Seit einiger Zeit gibt es zudem Displayauflösungen, die noch mehr Raum für Inhalte zur Verfügung stellen. Zu diesem Zweck wird das gleiche Layout in den Ordner *layout-xlarge* kopiert.

Nachdem das Interface implementiert wurde, dass das Detail-Fragment anpasst und die Activity des Modus Kontakte für den Einsatz eines Zwei-Spalten-Layouts erweitert, wird dieses Prinzip auf die Module News, Stundenplan und Notenanzeige übertragen.

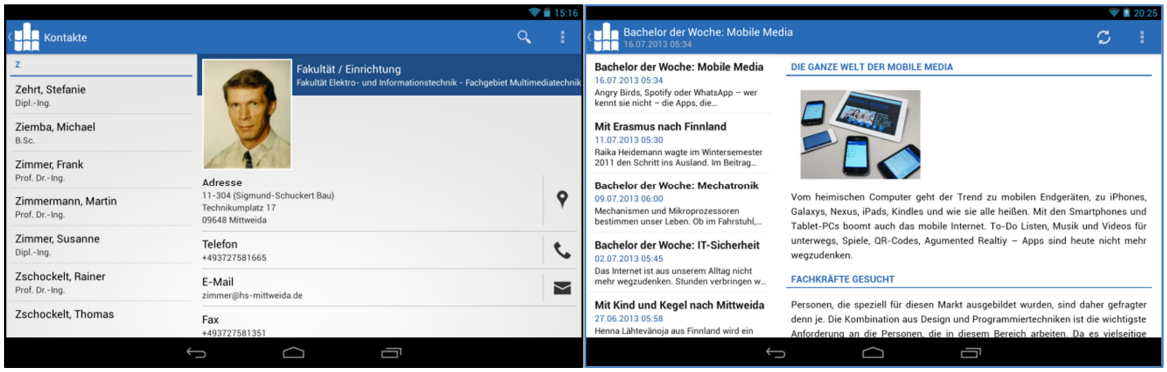

Abbildung 50 Landscape-Modus von Kontakte und News auf Tablets

Dazu werden die jeweiligen Layout-Dateien angelegt, ein Interface je Modul innerhalb des List Adapters implementiert, sowie die Detail-Fragments angepasst und die Activities erweitert. Im Modul Stundenplan muss zusätzlich der Aufruf des Interfaces an zwei verschiedenen Stellen getätigt werden, da innerhalb des Moduls verschiedene Views bereitgestellt werden, die jeweils die Detailansicht öffnen können.

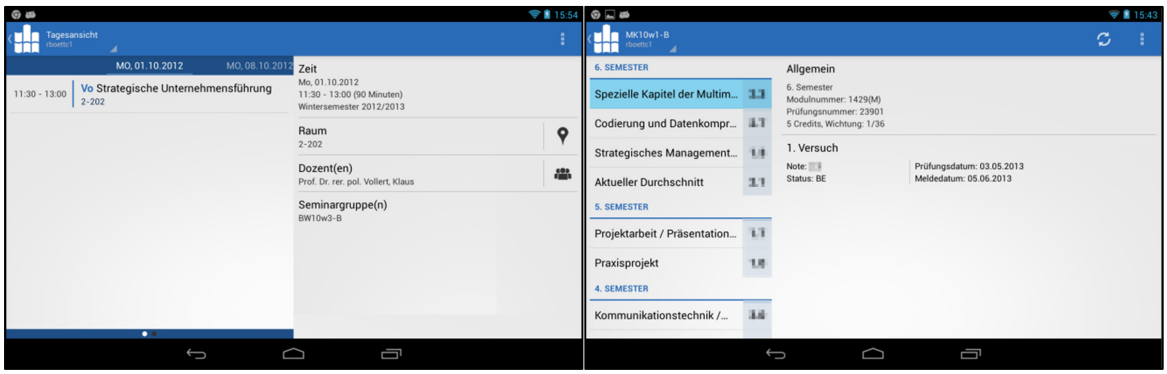

Abbildung 51 Landscape-Modus von Stundenplan und Notenanzeige auf Tablets

4.5 Weitere Anpassung

Die Umsetzung des Zwei-Spalten-Layouts passt die App optimal auf Bildschirmgrößen von Tablets an und bietet dem Benutzer eine intuitive Bedienung für den Umgang mit Listen- und Detailansichten, indem diese vereint innerhalb einer View dargestellt werden. Die View wird durch die Optimierung aufgeräumt und wirkt strukturiert. Allerdings kann dieses Prinzip nicht auf alle Module angewandt werden, da nicht in jedem Modul die Struktur von Listen- und Detailansicht vorhanden ist.

Beispielsweise ist das Zwei-Spalten-Layout nicht auf das Hauptmenü von HSMWmobil anwendbar, da dieses die einzelnen Module aufruft und es wenig Sinn machen würde, das Hauptmenü als Liste darzustellen und die Module als Fragments rechts neben dem Hauptmenü zu öffnen. Dieses Konzept wäre weder strukturiert, noch stimmig und würde die Benutzer wahrscheinlich verwirren, da das Hauptmenü und die Module nicht miteinander verknüpft werden sollten. Das gleiche Problem tritt im Modul Mensaplan auf, dass dem Benutzer zwar eine Listendarstellung aller Menüs der jeweiligen Tage liefert, jedoch keinerlei Detailansicht besitzt. Aus diesen Gründen muss ein separates Konzept gefunden werden, um solche Module ebenfalls an das größere Platzangebot von Tablets anzupassen und dieses sinnvoll auszunutzen.

Da es sich beim Mensaplan um eine Art Speisekarte handelt, lag die Vermutung nahe, das Layout so zu wählen, dass beiden Seiten des Mensaplans, die jeweils eine ganze Woche für die aktuelle, sowie folgenden Woche als Listenansicht darstellen, nebeneinander platziert werden können. Dieses Layout soll dabei eine Speisekarte nachbilden, wie sie in der Realität ebenfalls vorkommt. Der Vorteil dieses Konzeptes ist, dass der Benutzer somit den Eindruck erhält, dass er auf eine reale Speisekarte blickt. Die Umsetzung ist auf Grund der vorhandenen Fragments mit wenig Aufwand zu bewältigen. Des Weiteren ergibt diese Darstellung der Inhalte Sinn, da alle möglicherweise benötigten Informationen auf einen Blick verfügbar sind und das Platzangebot effizient ausgenutzt werden kann.

Für das Hauptmenü wird dieses Konzept ebenfalls übernommen, da es sich dabei um drei Seiten des View Pagers handelt, welches entsprechend der Kategorie die Module beziehungsweise deren Activity öffnen. Eine Einschränkung wird allerdings im Hauptmenü integriert, die die jeweiligen Fragments des Hauptmenüs lediglich im Landscape-Modus auf Tablets nebeneinander platziert. Begründet wird diese Entscheidung mit dem geringeren Platzangebot im Portrait-Modus auf Tablets, dass in der Breite für drei nebeneinander platzierte Fragments nicht ausreichend zur Verfügung steht. Innerhalb des Portrait-Modus soll in Zukunft auch auf Tablets der View Pager angezeigt und genutzt werden, um die jeweiligen Module zu öffnen.

Die Umsetzung setzt dabei voraus, dass die Activities bereits als Fragments vorhanden sind, was jedoch durch den View Pager bereits vorgegeben ist, weshalb die Umsetzung nur wenig Zeit in Anspruch nehmen sollte. Am Beispiel des Mensaplans wird innerhalb dieses Abschnittes die Umsetzung ausführlich erläutert und später im Hauptmenü ebenfalls angewandt.

Das Prinzip orientiert sich dabei am Prinzip des Zwei-Spalten-Layouts, greift jedoch nicht so tief in die Fragments ein, sondern ruft diese nur auf. Der Rest der Verarbeitung wird hierbei von den Fragments selbst übernommen. Des Weiteren sind die Fragments voneinander unabhängig und greifen nicht aufeinander zu.

Als erster Schritt wird ein neues Layout für die Anzeige der *MensaPlanActivity* erzeugt, welches zwei parallele Frame Layouts für die Anzeige der beiden einzelnen Fragments in einem Linear Layout mit dem Attribut *android:orientation* und dem Wert *horizontal* enthält. Das XML-Dokument für das Layout wird im Falle des Mensaplans in den Ordner *layout-large* erzeugt und in *layout-xlarge* später kopiert. Die beiden Frame Layouts werden jeweils in ein Linear Layout verschachtelt, welches das Attribut *android:orientation* mit dem Wert *vertical* erhält. Zusätzlich wird vor dem Frame Layout ein Text View hinzugefügt, das den Titel der jeweiligen Woche beinhaltet und die Attribute *android:background*, *android:gravitiy*, *android:text*, *android:padding* erhält. Die Hintergrundfarbe wird dabei vom View Pager entnommen und der Text mittig innerhalb des Text Views platziert. Die Linear Layouts erhalten das Attribut *android:layout_weight* und den gleichen Wert für ihre jeweilige Gewichtung. Dieses Attribut bewirkt, dass die beiden Fragments dasselbe Platzangebot erhalten. Zusätzlich erhalten beide Frame Layouts noch eine eindeutige ID für die spätere Zuordnung der Fragments. Das Layout im Standardordner für Layouts wird nicht verändert, da in diesem Fall der View Pager bestehen bleiben soll.

Der zweite und letzte Schritt umfasst umfangreiche Änderungen an der Activity des Moduls Mensaplan, da diese entscheidend dafür ist, welche View – View Pager oder die zwei Fragments – angezeigt werden soll. Dazu wird die Methode *displayContent(boolean update)* der Activity hinzugefügt, welche entweder die Fragments zu den jeweiligen Frame Layouts hinzufügt oder den View Pager instanziiert. Der Parameter *update* wird genutzt, um dem Download-Thread mitzuteilen, ob das komplette XML-Dokument heruntergeladen werden soll oder gegebenenfalls auf zwischengespeicherte Werte zurückgegriffen werden kann. Dieser Parameter wird beim Aufruf der Aktion *Aktualisieren* des Action Buttons benötigt, da in diesem Fall das XML-Dokument vollständig heruntergeladen werden soll um das vorhandene Dokument zu ersetzen. Innerhalb der Methode werden zuerst die Frame Layouts initialisiert und mit Hilfe der Methode *findViewById()* instanziiert. Eine *if*-Abfrage prüft anschließend, ob die Objekte nicht dem Wert *null* entsprechen und somit geladen sind. Auf Smartphones würde der Wert *null* zutreffen, weshalb somit die Gerätegruppe überprüft wird.

Entsprechend dieser Logik werden innerhalb der *if*-Abfrage die Fragments erstellt und später hinzugefügt. Im *else*-Zweig wird der View Pager instanziiert und aufgerufen.

```
protected void displayContent(boolean update) {
        Frame Layoutt thisFrame = (Frame Lay-
outt)findViewById(R.id.this_week_fragment_container);
        Frame Layoutt nextFrame = (Frame Lay-
outt)findViewById(R.id.next_week_fragment_container);

        if (thisFrame != null && nextFrame != null) {
                MensaPlanFragment thisWeek = createFragment(0, update);
                MensaPlanFragment nextWeek = createFragment(1, update);
                makeTransaction(thisWeek, nextWeek);
        } else {
                createViewPager(update);
        }
}
```

Abbildung 52 Methode *displayContent()* in der Activity des Moduls Mensaplans

Zu diesem Zweck wird eine weitere Methode *createFragment(int type, boolean update)* implementiert, die ein neues Fragment, sowie ein Bundle, instanziiert und das Fragment als Rückgabewert besitzt. Dem Bundle werden der *int*-Wert *type* und der boolesche Wert *update* hinzugefügt, der später vom Fragment verarbeitet werden. Das Bundle wird schließlich mit Hilfe der Methode *setArguements()* an das Fragment übergeben. Anhand von *type* wird festgelegt, ob es sich um die aktuelle oder die folgende Woche für die Darstellung des Speiseplans handelt. Dahingegen wird *update* direkt durch den Methodenaufruf an die Methode *createFragment(int type, boolean update)* weitergegeben.

```
protected MensaPlanFragment createFragment(int requestType, boolean update) {
        MensaPlanFragment fragment = new MensaPlanFragment();
        Bundle args = new Bundle();
        args.putInt("pos", requestType);
        args.putBoolean("update", update);
        fragment.setArguments(args);
        return fragment;
}
```

Abbildung 53 Methode *createFragment()* in der Activity des Mensaplans

Des Weiteren wird eine weitere Methode *makeTransaction(MensaPlanFragment frag1, MensaPlanFragment frag2)* zur Activity hinzugefügt, welche unmittelbar nach dem Erstellen der Fragments innerhalb von *displayContent(boolean update)* aufgerufen wird, um die Fragments zu den Frame Layouts hinzuzufügen. Die vorausgesetzten Parameter dieser Methode entsprechen den beiden Fragments für die aktuelle und folgende Woche des Mensaplans. Diese werden jeweils innerhalb von *displayContent(boolean update)* und mit Hilfe der Rückgabewerte von *createFragment(int type, boolean update)* instanziiert.

```
protected void makeTransaction(MensaPlanFragment thisWeek, MensaPlanFragment nextWeek) {
        FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
        transaction.replace(R.id.mensaplan_this_week_fragment_container, thisWeek);
        transaction.replace(R.id.mensaplan_next_week_fragment_container, nextWeek);
        transaction.commit();
}
```

Abbildung 54 Methode *makeTransaction()* in der Activity des Moduls Mensaplan

Die Methode *makeTransaction(MensaPlanFragment frag1, MensaPlanFragment frag2)* instanziiert dabei eine neue Fragment Transaction und fügt die, über die Parameter übergebenen, Fragments mit Hilfe der Methode *replace()* der Fragment Transaction hinzu. Schlussendlich wird die Fragment Transaction durch den Aufruf der Methode *commit()* übergeben und durchgeführt. Somit werden die Fragments mit Hilfe der Fragment Transaction den Frame Layouts hinzugefügt und existieren innerhalb der Activity.

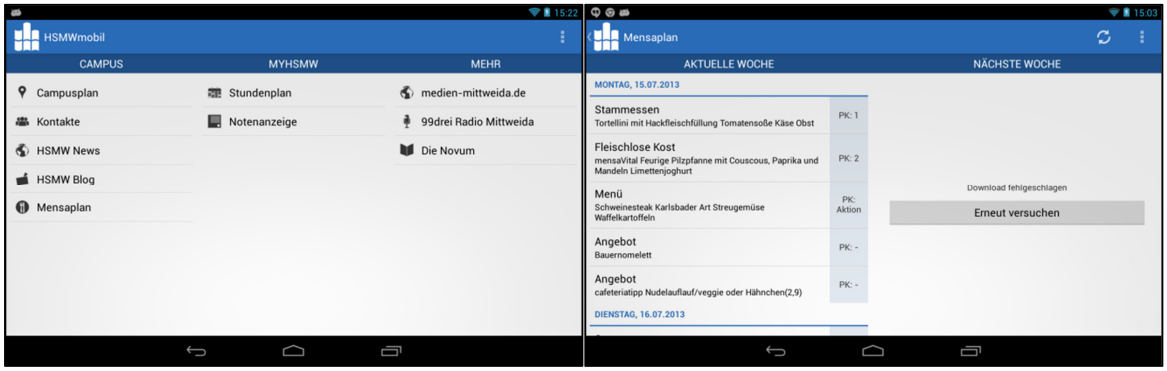

Abbildung 55 Landscape-Modus des Hauptmenüs und Mensaplan auf Tablets

Für den Fall, dass ein Smartphone die Activity aufruft und somit die Frame Layouts nicht vorhanden sind, wird der View Pager instanziiert. Diesen übernimmt die Methode *createViewPager(boolean update)*, die im *else*-Zweig der Methode *displayContent(boolean update)* aufgerufen wird. Innerhalb der Methode werden der Pager Tab Strip, ein Pager Adapter und ein Page Indicator Builder erstellt, wobei der Pager Tab Strip die Anzeige des aktuellen Fragments übernimmt und am oberen Rand des Layouts angezeigt wird. Wird zwischen den Fragments gewechselt, ändert sich der entsprechende Titel. Der Pager Adapter übernimmt das Hinzufügen von Views, sowie die Beschriftung der Titel des Pager Tab Strips. Des Weiteren wird der View Pager initialisiert und mit Hilfe der Methode *findViewById()* instanziiert. Der Pager Adapter wird anschließend dem Objekt des View Pagers hinzugefügt.

```
protected void createViewPager(boolean update) {
        PagerTabStrip tabStrip = (PagerTabStrip) findViewById(R.id.pager_tab_strip);
        tabStrip.setDrawFullUnderline(true);
        tabStrip.setTabIndicatorColor(getResources().getColor(R.color.viewpager_underline));

        MensaPlanPagerAdapter mPageAdapter = new MensaPlanPagerAdapter(
                getSupportFragmentManager(), update);
        mViewPager = (ViewPager) findViewById(R.id.viewpager);
        mViewPager.setAdapter(mPageAdapter);

        PageIndicatorBuilder pageIndicator = new PageIndicatorBuilder(
                getApplicationContext(), mViewPager);
        baseLayout = (LinearLayout) findViewById(R.id.viewpager_layout);
        baseLayout.addView(pageIndicator.getIndicator());
}
```

Abbildung 56 Methode createViewPager() der Activity des Moduls Mensaplan

Der Page Indicator Builder greift auf den View Pager zu, um die Anzahl der beinhaltenden Fragments zu ermitteln und entsprechend ausreichend Kreise für die Anzeige des aktuellen Fragments hinzufügen zu können und das aktuelle Fragment zu markieren. Des Weiteren wird ein Linear Layout für dessen Darstellung initialisiert und die View des Page Indicator Builders hinzugefügt. Somit wird auf Smartphones weiterhin der View Pager aufgerufen und auf Tablets auf das neue Layout, und somit auch auf die neue Darstellung, zurückgegriffen.

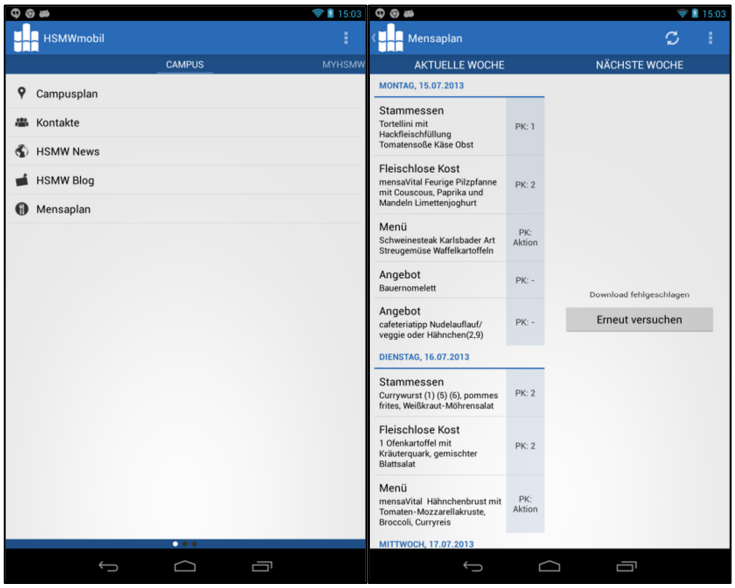

Abbildung 57 Portrait-Modus von Hauptmenü und Mensaplan auf Tablets

Auch für das Hauptmenü wird nach diesem Verfahren gearbeitet und die Fragments lediglich im Landscape-Modus nebeneinander platziert. Dazu muss das Layout im Gegensatz zum Mensaplan nicht im Ordner *layout-large*, sondern im Ordner *layout-large-land* platziert werden. Die weiteren Arbeitsschritte unterscheiden sich nicht weiter von denen im Mensaplan.

4.6 Probleme und Lösungen

4.6.1 Fehlerarten

Während der Programmierung kann es häufig zu Problemen kommen. Diese können unterschiedliche Ursachen haben, so dass es schwierig sein kann, Probleme schnell zu lösen. Ein wichtiges Werkzeug für die Lokalisierung und für die spätere Behebung von Problemen ist dabei das LogCat von Eclipse, welches Exceptions auf der Konsole sichtbar macht und mit Hilfe des Debuggers von Eclipse den Auftritt eines Fehlers lokalisiert. Auf der Konsole werden die Datei und die entsprechende Zeile angezeigt, in der der Fehler auftritt und die App abbricht. Fehler lassen sich klassifizieren, da neben den Syntaxfehlern auch weitere Fehler, wie:

- Lexikalische Fehler
- Semantische Fehler
- Logische Fehler
- Laufzeitfehler

existieren. Die meisten Fehler werden von Eclipse bereits in Echtzeit während der Programmierung angezeigt. Bei logischen Fehlern und Laufzeitfehlern ist dies jedoch nicht möglich, da Probleme während der Laufzeit eines Programms sporadisch und unerwartet auftreten können.

Unter Java und somit auch in der Android-Entwicklung existieren sogenannte Ausnahmen – die Exceptions. Darunter sind Fehler zu verstehen, die erst zur Laufzeit eines Programms auftreten und zum Absturz des Programms führen, wenn sie nicht abgefangen werden. Die häufigste Exception stellt dabei die *NullPointerException* dar, welche immer auftritt, wenn auf ein Objekt zugegriffen wird, dass *null*, also leer, ist. Der Debugger zeigt den Fehler in der Regel an der Stelle an, an der er auftritt. Es ist jedoch möglich, dass Fehler tief verschachtelt sind, wenn beispielsweise über mehrere Klassen und Hilfsklassen hinweg gearbeitet wird. Die Ursachenforschung kann sich in diesem Fall zeitlich in die Länge ziehen, da der Entwickler den Fehler manuell nachverfolgen und beheben muss.

Die sogenannte Rechtschreibung eines Quelltextes stellt unter Eclipse keinerlei Problem dar, da der Quelltext in Echtzeit durch den Compiler überprüft wird und beispielsweise lexikalische oder semantische Fehler sofort mit einem Hinweis in der jeweiligen Zeile angezeigt und markiert werden.

4.6.2 Entwicklungsumgebung Eclipse

Auch die Entwicklungsumgebung Eclipse hat ihre Tücken und hat während der Umsetzung für Probleme gesorgt. Nach einem Update des ADT-Plug-Ins von Google auf die aktuellste Version, zeigte Eclipse verschiedene Fehler innerhalb des Quelltextes an, die vorher nicht vorhanden waren, obwohl keine Änderungen vorgenommen wurden. Eine Analyse der Fehler hat ergeben, dass die Fehler auf Grund einer nicht mehr ordnungsgemäßen Integration der Bibliothek Google Play Services aufgetreten sind. Um das Problem zu beheben, wurde eine aktualisierte Version der Bibliothek mit Hilfe des SDK Managers von Android in Eclipse heruntergeladen und installiert. Des Weiteren wurden alle Importe und Verweise auf die Bibliothek entfernt und neu hinzugefügt. Nach dem Aktualisieren der Importe wurden die Fehler nicht mehr angezeigt und Eclipse war wieder einsatzbereit. Die Ursache des Fehlers ist dabei in Inkompatibilitäten zwischen der Bibliothek und dem ADT-Plug-In zu suchen und konnte mit Hilfe des Updates behoben werden.

Auch die Integration von Testgeräten verursachte Fehler innerhalb von Eclipse. So verursachte der ADB-Treiber von Google für das Nexus 7 diverse Fehler, sodass das Gerät zwar erkannt, jedoch in Eclipse nicht aufgeführt war. Trotz aktiviertem USB-Debugging tauchte das Gerät in Eclipse nicht auf. Open Source-Treiber aus dem Internet konnten das Problem, trotz Verweis in diversen Foren[35], nicht lösen. Erst die Umstellung der USB-Verbindung in den Einstellungen des Nexus 7 von MTP[36] auf PTP[37] brachte den gewünschten Erfolg und das Nexus 7 konnte zum Testen und Debuggen genutzt werden.

[35] Vgl.: http://forum.xda-developers.com/showthread.php?t=1766220, 2013

[36] Erweiterung des PTP[37].

[37] Ermöglicht Übertragung von Dateien zwischen mobilen Geräten und Computern.

Ein weiterer Fehler der während der Entwicklung aufgetreten ist und durch die Verwendung von Eclipse verursacht wurde, war die Fehlermeldung „*Class not Found – ClassNotFoundException*“ beim Starten der App, obwohl die angegebene Klasse vorhanden war. Nach gründlicher Recherche und Ratlosigkeit über die Herkunft des Fehlers, wurde festgestellt, dass es sich bei diesem Fehler um einen Fehler des Class Loaders handelte. Da dieses Problem anscheinend durch fehlerhafte Verweise von Ressourcen und Klassen entstanden ist, wurden diese gesichert, das Projekt gelöscht und neu aus dem SVN der Hochschule Mittweida heruntergeladen. Da jedoch der Entwicklungsstand wieder auf Version 1.1 beruht, mussten die gesicherten Klassen und Ressourcen wieder eingegliedert werden. Nach der Wiederherstellung des Projektes wurde die Versionsnummer und -beschreibung im Manifest angehoben und das Projekt konnte getestet werden.

Letzteres Problem mit Eclipse trat innerhalb des Projektes mehrfach auf, so dass das Projekt mehrfach wieder neu gebaut werden musste. Die Ursache hierfür konnte schlussendlich nicht gänzlich ermittelt werden, aber die Lösung angewandt werden.

4.6.3 Action Bar

Die Split Action Bar wurde in den Campusplan implementiert und integriert, bereitet an einer Stelle jedoch Probleme, da die Methoden *onCreateOptionsMenu(Menu menu)* und *onOptionsItemSelected(MenuItem item)* keine Anzeige der Action Buttons innerhalb der Bottom Bar bewirken. Die Ursache des Problems ist dabei der API Level von HSMWmobil. Der Menu Inflater arbeitet hierbei nicht korrekt, wodurch die Menü-XML nicht referenziert wird und keine Items gezeichnet werden können.

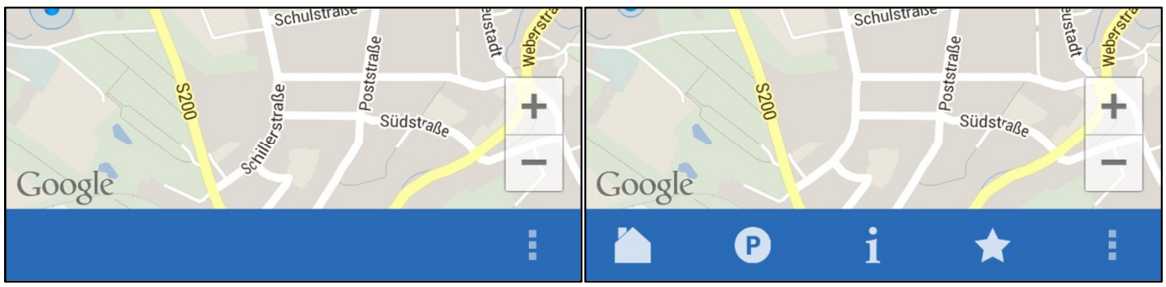

Abbildung 58 Fehlende Action Buttons in der Split Action Bar

Um das Problem zu beheben, muss das Map-Fragment die Methoden *OnCreateOptionsMenuListener* und *OnOptionsItemSelectedListetner* implementieren, sowie die korrekten Verweise auf die Bibliotheken eingetragen werden. Dadurch ist es möglich den Menu Inflater von ActionBarSherlock zu verwenden, welcher den Inflater der Menü-XML und die Anzeige der Action Buttons korrekt vornimmt.

4.6.4 Campusplan

Ein weiteres Problem betrifft den Campusplan, der auf Fragments umgestellt wurde. Ursprünglich war geplant, dass der Campusplan ähnlich den Modulen Kontakte, News, Stundenplan und Notenanzeige aufgebaut sein sollte. Es sollte in der Landscape-Ansicht auf der linken Seite der Campusplan eingeblendet werden und auf der rechten Seite die Detailansicht eingeblendet werden, sobald ein Gebäude ausgewählt wird. Dabei sollte das Detail-Fragment dynamisch erzeugt und eingeblendet werden, nachdem ein Gebäude ausgewählt wurde. Da die Fragments jedoch den gesamten Platz in Anspruch nehmen und nicht nebeneinander zu einem Container hinzugefügt werden können, musste ein anderer Weg gefunden werden. Eine Überlegung dazu war ein weiteres Frame Layout neben dem eigentlich Frame Layout für das Map Fragment innerhalb des Layouts hinzuzufügen und mit dem Attribut *android:layout_weight* und dem Wert *0* zu belegen. Nachdem ein Gebäude aufgerufen wurde, sollte im Quelltext die Gewichtung entsprechend geändert werden. Jedoch funktionierte dieses Vorgehen nicht wie gewünscht, sodass beschlossen wurde, den Campusplan so zu belassen, wie er ist und lediglich die Detailansicht zu optimieren, wenn sich die View in der Portrait-Ansicht befindet. Dieser Beschluss hat auch keine weiteren Folgen, da das Map-Fragment so die komplette View ausfüllt, was wiederum dem Benutzer zu Gute kommt, da die Karte ausreichend groß ist.

Da die Detailansicht jedoch unpassend wirkt, wird diese auf die Anzeige auf Tablets in der Landscape-Ansicht optimiert, indem Elemente des Layouts nebeneinander platziert und dargestellt werden.

4.6.5 Sicherung und Wiederherstellen einer Web-View im Lifecycle

Wenn eine Web View innerhalb einer Activity pausiert oder gestoppt wird, tritt ein weiteres Problem auf. Dieser Fall tritt beispielsweise auf, wenn das Gerät gedreht werden soll und die Activity von dem Portrait-Modus in den Landscape-Modus oder andersherum wechselt. Wird beispielsweise in der Web View gescrollt, muss die aktuelle Position gespeichert werden, um nach dem Drehen des Gerätes an der gleiche Stelle mit dem Lesen fortfahren zu können.

Um den Zustand der Web View zu speichern, muss die Methode *onSaveInstanceState()* um den Aufruf *mWebView.saveState(outState)* erweitert werden. Durch die Erweiterung speichert die instanziierte Web View ihren Zustand im Bundle *outState* der Methode. Die Wiederherstellung des Zustandes aus dem Bundle wird wiederum innerhalb der Methode *onCreate()* durchgeführt, indem das Bundle *savedInstanceState* mit dem Wert *null* verglichen wird. Ist das Bundle nicht *null*, wird der Zustand mit Hilfe der Methode *restoreState()* und dem Bundle *savedInstanceState* als Parameter wiederhergestellt. Dazu muss an dieser Stelle die Web View referenziert werden und die Methode über die Referenz aufgerufen werden.

Da innerhalb von *onStart()* jedoch die Detail-Ansicht aktualisiert wird und im Falle der Web View die URL mit Hilfe der Web View neu geladen wird, muss die Methode *updateDetailView(int pos, Bundle args)* um den booleschen Parameter *update* erweitert werden. Dieser Parameter soll die Methode darüber informieren, ob die Web View die URL neu laden muss oder ob die Web View aus *savedInstanceState* wiederhergestellt worden ist. Außer dem Fall, dass die Ansicht wiederhergestellt wird und der Wert *false* übertragen werden muss, wird in allen Fällen dem Methodenaufruf der Wert *true* übergeben.

Somit wird die Web View nicht neugeladen und der vorhandene Zustand beim Drehen des Displays aus dem Bundle wiederhergestellt. Der Benutzer ist somit in der Lage an der Stelle mit Lesen fortzusetzen, an dem er das Display gedreht hat.

4.6.6 Webservice

Auf Grund der Tatsache, dass einige Services, die der Webservice zur Verfügung stellt, weiterentwickelt oder überarbeitet werden, können Daten des Öfteren auf Grund von Nichtverfügbarkeit des Webservices nicht abgerufen werden.

Um das Problem zu lösen werden Daten der einzelnen Module gespeichert und mit Hilfe des Cloud-Services Dropbox[38] gesichert. Mit Hilfe von Dropbox können die Daten so dauerhaft über das Internet verfügbar gehalten werden. Diese Daten sind allerdings statisch, weswegen lediglich im Falle einer Nichtverfügbarkeit auf die gespeicherten Daten zurückgegriffen wird und im Normalfall die Live-Daten des Webservices verwendet werden, um aussagekräftige Testbedingungen zu simulieren.

4.7 Abschließende Tests

Eine ausführliche Teststrategie ist für umfangreiche Projekte stets von Vorteil. Des Weiteren sollten am Ende jedes Arbeitsprozesses Funktionstests durchgeführt werden, um auftretende Probleme schnell lokalisieren und letztlich auch lösen zu können. Die Tests sollen zum einen dazu dienen, feststellen zu können, ob die gesetzten Anforderungen erfüllt und die Qualität der Software ausreichend ist. Dabei sollen sowohl das Gesamtprojekt, als auch die einzelnen Module und Komponenten die eigenen und gestellten Anforderungen und Erwartungen erfüllen.

Um umfangreiche Tests durchführen zu können, muss die Testumgebung umfangreich aufgestellt sein. Zu diesem Zweck werden die privaten Testgeräte mit verschiedenen Android-Versionen ausgestattet. Auf dem ViewPad 10s werden nach und nach sowohl Android in Version 3.1, 4.0, 4.1 und auch Android in Version 4.2 installiert. Ein geöffneter Bootloader[39] und eine aktive Community machen solch umfassende Bedingungen möglich.

[38] Vgl.: https://www.dropbox.com/, 2013

[39] Startet den Boot-Vorgang und schlussendlich das Betriebssystem.

Für die Überprüfung der Funktionalität auf Smartphones werden auf dem Nexus 4 die Android-Version 4.1 und 4.2 eingespielt. Auf dem Sony Ericsson Arc S werden zusätzlich die Android-Versionen 2.3, 4.0, 4.1 und 4.2 der letzten Versionen von CyanogenMod[40] installiert. Die Geräte der Hochschule, speziell das Nexus 7, werden mit den jeweilig aktuell verfügbaren Versionen von Android für diese Geräte für die Tests bereitgestellt. Im Falle des Nexus 7 ist dies die Version 4.2 Jelly Bean.

Auf Grund der hohen Verbreitung und der zahlreichen Geräte, die mit Android ausgestattet sind, ist es unmöglich alle Bereiche abzudecken. Diese umfangreiche Anzahl an Testgeräte und Versionen von Android sollten allerdings eine gute Basis dafür bieten, die App auf möglichst vielen Geräten verfügbar zu machen. Dabei sollten ebenfalls viele Szenarien eintreten, die eine Veröffentlichung simulieren und somit mögliche Fehlerquellen frühzeitig aufzeigen.

Für die Tests sind die Testgeräte weitestgehend mit der Entwicklungsumgebung Eclipse verbunden um sofort debuggen zu können. Dazu werden alle Log-Dateien von Android mit Hilfe des Debuggers LogCat ausgelesen, um im Falle einer Exception sofort reagieren zu können. Über die Log-Dateien lässt sich zudem nachvollziehen, was gerade innerhalb der App passiert. Mit sinnvoll integrierten Ausgaben innerhalb des Quelltextes ist somit der gesamte App-Verlauf nachvollziehbar.

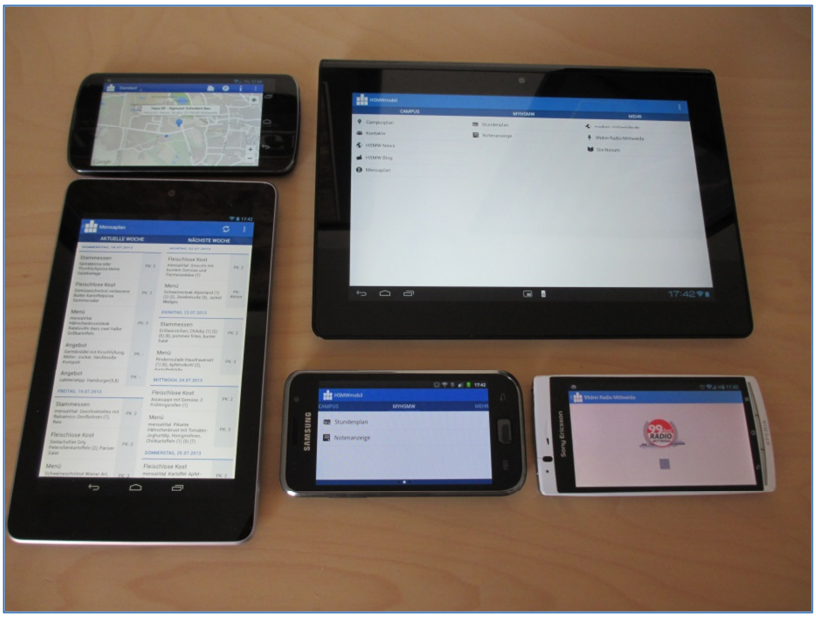

Abbildung 59 Testgeräte für das Bachelor-Projekt

[40] Vgl.: http://www.cyanogenmod.org/, 2013

Die Durchführung der Tests wird dabei im ersten Schritt vom Entwickler und Autor selbst übernommen. Dabei stehen vor allem Stabilität und Funktionalität im Vordergrund. Bemühungen die App konsistent gegenüber Fehler zu machen stehen ebenso auf der Agenda. Treten während dieser sogenannten Entwicklertests keine feststellbaren Fehler auf, werden Anwendertests mit ausgewählten Personen, die mit dem Projekt nichts zu tun haben, durchgeführt. Während dieser Anwendertests wird das gewöhnliche Verhalten von Benutzern der App simuliert und es ist möglich verschiedene unerwartete Szenarien einzuleiten und durchzuführen. Diese Anwendertests werden von einer kleinen Anzahl Personen durchgeführt und vom Entwickler überwacht.

Nach Abschluss der Anwendertests wird die aktuelle Entwicklungsversion den Projektbetreuern zur Verfügung gestellt und anschließend ausführlich getestet. Dieser sogenannte Abnahmetest bildet dabei die Grundlage für eine spätere Veröffentlichung des gesamten Projektes. Zum Abschluss des Projektes werden die aktuellen Entwicklungen aus dem Hauptzweig in das Projekt übernommen und das Projekt anschließend als Hauptprojekt deklariert.

5 Ausblick und Fazit

5.1 Ausblick

Mit Abschluss des Bachelor-Projektes ist die Entwicklung von HSMWmobil keineswegs abgeschlossen und weitere Entwicklungen sind bereits in Arbeit beziehungsweise geplant. Beispielsweise wird von den anderen beiden Entwicklern der Android-Version von HSMWmobil ein Modul integriert, welches Push-Nachrichten ermöglichen soll, um über neue Noten oder über Stundenplanänderung automatisch informiert zu werden. Eine weitere Entwicklung umfasst die Integration des Stundenplans in den Kalender von Android. Diese Entwicklung beinhaltet einen sogenannten Sync-Adapter und Änderungen am Accounting-System von HSMWmobil, welches die Anmeldedaten für den internen Bereich der App verwaltet.

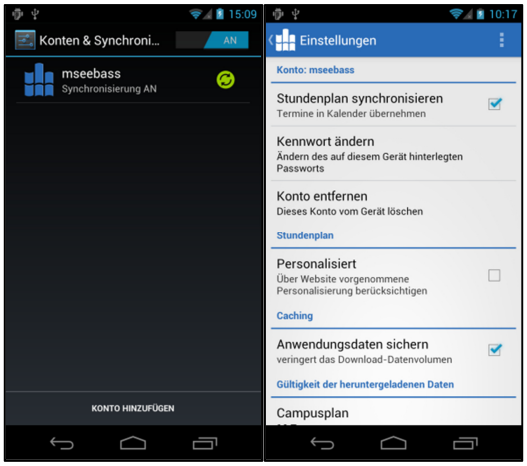

Abbildung 60 Sync-Adapter von HSMWmobil[41] unter Android

Ebenso wie diese Entwicklungen sollen die Detailansichten der einzelnen Module vereinheitlicht und neu gestaltet werden. Die Detailansichten sollen dabei übersichtlicher, funktioneller und aufgeräumter umgesetzt werden. Des Weiteren soll die Verknüpfung der Module untereinander weiter gefördert werden, um dem Benutzer die geforderten Informationen schnell und intuitiv zur Verfügung stellen zu können.

[41] Vgl.: Bildschirmfotos von Michael Seebaß

Eine Neugestaltung des Hauptmenüs ist ebenso denkbar, da dieses veraltet und nur bedingt benutzerfreundlich implementiert ist. Die Activity wird vollständig entfernt und die Navigation erfolgt ausschließlich über ein weiteres Konzept. Dazu könnte ein sogenannter Navigation Drawer[42] integriert werden, der neben dem App-Icon der Action Bar 3 übereinanderliegende Striche hinzufügt und ein Menü beim Betätigen des App-Icon über der aktuellen Activity anzeigt. Innerhalb des Menüs könnten die einzelnen Module aufgelistet und aufgerufen werden, was der Funktionalität eines Hauptmenüs entspricht.

Der Navigation Drawer ist dabei in allen Modulen in der höchsten Navigationsebene verfügbar. Beim Navigieren innerhalb der Hierarchie wird der Navigation Drawer ausgeblendet und die Standardelemente der Action Bar sind dabei für die Navigation abrufbar. Es ist jedoch möglich über das App-Icon zur höchsten Hierarchieebene zu navigieren.

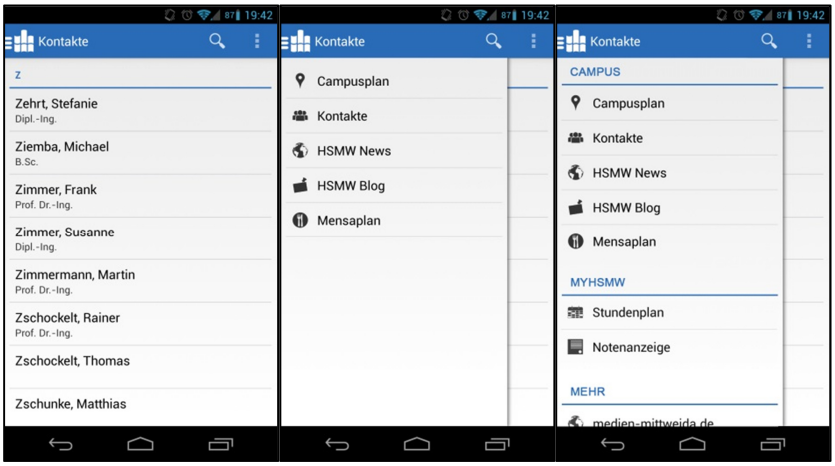

Abbildung 61 Konzeptzeichnung für den Navigation Drawer von Android

Zudem ist es möglich die Navigationselemente in Gruppen zu gliedern, sodass die einzelnen Fragments im Prinzip eins zu eins nachgebildet werden können. Dabei werden die Gruppen Campus, MyHSMW und Mehr mit den jeweiligen Einträgen dargestellt. Der einzige Unterschied ist die Positionierung der Gruppen untereinander.

Auch die Umsetzung der einschiebenden Detailansicht im Campusplan wird mit Abschluss des Projektes nicht verworfen und wäre eine mögliche weitere Anpassung in zukünftigen Updates. Diese und viele weitere Konzepte sind denkbar, um das Benutzerempfinden weiter zu steigern, sowie die App stetig zu verbessern und zu erweitern. Welche davon in naher oder ferner Zukunft umgesetzt werden, oder welche nicht den Weg in HSMWmobil finden, ist an dieser Stelle nicht abschätzbar. Allerdings wird die Entwicklung stetig vorangetrieben und fortgesetzt.

[42] Konzept von Google für die Gestaltung eines Menüs.

5.2 Fazit

Das Ziel des Projektes war die weitest gehende Optimierung der Oberfläche der App HSMWmobil der Hochschule Mittweida für Tablets. Dabei sollten die bestehenden Module entsprechend verändert und überarbeitet werden, um das zusätzliche Platzangebot sinnvoll zu nutzen und dem Benutzer ein einheitliches, sowie plattformtreues Empfinden zu gewährleisten. Mit dem Abschluss des Bachelor-Projektes wurden die gestellten Anforderungen vollständig erfüllt, sowie eine Grundlage für die weitere Entwicklung von HSMWmobil geschaffen. Es wurden sowohl die Kernziele, also auch die nebenläufigen Ziele, welche mit der Umsetzung der Optimierung gefordert waren, umgesetzt. Die jeweiligen Module nutzen den Platz effizient aus und das Zwei-Spalten-Layout bietet neben der besseren Übersichtlichkeit die Möglichkeit mehr Informationen dem Benutzer zugänglich zu machen. Im Großen und Ganzen kann das Projekt als gelungen angesehen werden, da alle Widrigkeiten, die mit der Umsetzung einhergingen weitestgehend beseitigt werden konnten und die Optimierung vollständig vollzogen werden konnte.

Persönlich hat mir die Umsetzung des Projektes viel Spaß gemacht und ich konnte meine Kenntnisse im Bereich der Android-Entwicklung weiter vertiefen und festigen. Meine Kenntnisse und Fähigkeiten konnte ich im Laufe des Praktikums stets einbringen und anwenden. Zudem konnte ich meine Kenntnisse um einen interessanten und aktuellen Themenbereich erweitern. Besonders interessant war die alleinige Abarbeitung des gesamten Projektes mit definierter Zielvorgabe. Mit Abschluss des Projektes steht der Hochschule Mittweida ein einheitliches Produkt für die Smartphone- und Tablet-Bereiche von Android zur Verfügung.

Zusammenfassend bin ich mit meiner Leistung, sowie dem Gesamtergebnis zufrieden. Ich bin mir zudem sicher, dass mir die gewonnenen Erfahrungen, Kenntnisse und Fähigkeiten im Laufe der weiteren Studien- und Arbeitszeit zu Gute kommen werden. Ich kann mir vorstellen, in Zukunft weitere Projekte im Bereich der Android-Entwicklung zu übernehmen und nach Abschluss des Studiums eine Beschäftigung in diesem Bereich aufzunehmen.

Literatur

[andc2013] androidcentral, Jerry Hildenbrand:
IDC: Android now leads the tablet market with a 56,5% share.
URL: http://m.androidcentral.com/idc-android-now-leads-tablet-market-565-share, verfügbar am 05.01.2013, aufgerufen am 04.06.2013

[DACT2013] Google Android Developers:
Activities.
URL:
http://developer.android.com/guide/components/activities.html, aufgerufen am 24.06.2013

[DBLO2008] Android Developers Blog:
Touch Mode.
URL: http://android-developers.blogspot.mx/2008/12/touch-mode.html, verfügbar am 01.12.2008, aufgerufen am 07.05.2013

[DDYN2013] Google Android Developers:
Building a Dynamic UI with Fragments.
URL:
http://developer.android.com/training/basics/fragments/index.html, aufgerufen am 07.05.2013

[DFRA2013] Google Android Developers:
Fragments.
URL:
http://developer.android.com/guide/components/fragments.html, aufgerufen am 05.05.2013

[DFRA2013] Google Android Developers:
Supporting Multiple Screens.
URL:
http://developer.android.com/guide/practices/screens_support.html , aufgerufen am 05.05.2013

[DMPL2013] Google Android Developers:
Multi-pane Layouts.
URL: http://developer.android.com/design/patterns/multi-pane-layouts.html, aufgerufen am 05.05.2013

[DRES2013] Google Android Developers:
More Resource Types.
URL: http://developer.android.com/guide/topics/resources/more-resources.html#Dimension, aufgerufen am 04.07.2013

[DSET2013] Google Android Developers:
Setup Google Play Services SDK.
URL: http://developer.android.com/google/play-services/setup.html, aufgerufen am 06.05.2013

[DTAS2013] Google Android Developers:
Tasks and Back Stack.
URL: http://developer.android.com/guide/components/tasks-and-back-stack.html, aufgerufen am 12.05.2013

[gole2013] golem, IT-News für Profis; Ingo Pakalski:
Android überholt iOS in diesem Jahr.
URL: http://www.golem.de/news/tablets-android-ueberholt-ios-in-diesem-jahr-1303-98167.html, verfügbar am 13.03.2013, aufgerufen am 04.06.2013

[GROK2013] Grokking Android; Wolfram Rittmeyer:
Adding ActionBar Items From Within Your Fragments.
URL: http://www.grokkingandroid.com/adding-action-items-from-within-fragments/, verfügbar 17.01.2013, aufgerufen am 08.05.2013

[IDCC2013] IDC:
Android and iOS Combine for 92.3% of All Smartphone Operating System Shipments in the First Quarter While Windows Phone Leapfrogs BlackBerry, According to IDC.
URL: http://www.idc.com/getdoc.jsp?containerId=prUS24108913, verfügbar am 16.05.2013, aufgerufen am 04.06.2013

[REXR2013] Mapsaurus:
PanesLayout: built multi-pane tablet layouts in Android.
URL: http://blog.mapsaurus.com/post/43459861627/paneslayout-build-multi-pane-tablet-layouts-in-android, verfügbar am 18.02.2013, aufgerufen am 10.06.2013

[SHER2012] Xavi Rigau:
HowTo: ActionBarSherlock + MapFragment + ListFragment.
URL: http://xrigau.wordpress.com/2012/03/22/howto-actionbarsherlock-mapfragment-listfragment/, verfügbar am 22.03.2012, aufgerufen am 24.05.2013

[SHER2013] ActionBarSherlock; Jake Wharton:
ActionBarSherlock: Usage.
URL: http://actionbarsherlock.com/usage.html, aufgerufen am 06.05.2013

[SPIE2011] SPIEGEL ONLINE, Netzwelt:
Googles Handy-Betriebssystem überholt Nokia.
URL: http://www.spiegel.de/netzwelt/gadgets/marktforscher-googles-handy-betriebssystem-ueberholt-nokia-a-742674.html, verfügbar am 31.01.2011, aufgerufen am 04.06.2013

[STOC2013] Stackoverflow, Questions; Jovy, Dmitry Zaitsev, Jordy Langen:
ClassNotFoundException reading a Serializable object in a class extending MapFragment in onSaveInstanceState.
URL:
http://stackoverflow.com/questions/14612336/classnotfoundexception-reading-a-serializable-object-in-a-class-extending-mapfra, verfügbar am 31.01.2013, aufgerufen am 07.07.2013

[STOD2013] Stackoverflow, Questions; Padma Kumar, Rajdeep Dua:
Design Layout For Multiple Screens.
URL: http://stackoverflow.com/questions/8428096/design-layout-for-multiple-screens, verfügbar am 08.12.2011, aufgerufen am 07.05.2013

[ZDNe2013] ZDNet, Mobil; Björn Greif:
Gartner: Android erreicht fast 75 Prozent Marktanteil.
URL: http://www.zdnet.de/88154889/gartner-android-erreicht-fast-75-prozent-marktanteil/, verfügbar am 14.05.2013, aufgerufen am 04.06.2013

Anlagen

Beiliegendes Medium:

- Projekt: HSMWbachelor
- Projekt: ActionBarSherlock (Bibliothek)
- Projekt: google-play-services (Bibliothek)

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 26. Juli 2013

René Böttcher